Copyright Information

Disclaimer

Table of Contents

Security

Note

Load the *Security* initial configurations prior to starting working on the tasks.

11.1 AAA Authentication Lists

- Configure R1 to use a TACACS+ server at the IP address 155.X.146.100 and encrypt communications using a key value of CISCO.
- Source TACACS+ packets from the Loopback0 interface.
- Configure the router so that access to the console line uses local authentication.
- Ensure the VTY lines authenticate users with TACACS+ and then fall back to line authentication.
- Privilege mode authentication should first attempt TACACS+ and then fall back to the local password.
- Create a user named "ADMIN" with a password of "CISCO" in the local database for these configurations.
- Customize the prompts for AAA user authentication and change the default banner message.

11.2 AAA Exec Authorization

- Authorize exec privilege levels using the remote TACACS+ server.
- Ensure that console line logins are also authorized. The console line should authorize users with the TACACS+ method first and then the local database configuration.
- User "ADMIN" should be granted a privilege level of 7 if authenticated.
- Use the VTY line settings to authorize any authenticated users in the event the TACACS+ server fails.
- Ensure that if the TACACS+ server is unavailable, authenticated users logging in via VTY lines are placed in privilege level 15.

11.3 AAA Local Command Authorization

- Ensure that the user “ADMIN” can use RIP debugging commands and can disable any currently active debugging using a single command.
- The same user should be able to configure any interface IP settings and administratively enable or disable any of these interfaces.
- Ensure the user can see their permitted commands in their running configuration.

11.4 Traffic Filtering with Standard Access-Lists

- Using the minimum amount of ACL lines, allow access to R1 from the Loopback0 subnets of R2 through R6 that have an even 3rd octet.
- Deny access from any other “150.X.0.0/16” subnets.
- Any hosts from the “155.X.0.0/16” subnets should still be able to reach R1.
- Apply the access-list inbound on all interfaces of R1.

11.5 Traffic Filtering with Extended Access-List

- Configure R4 to permit any TCP traffic destined for R5 and sourced from the Loopback0 subnets. Ensure R4 allows returning TCP packets for those connections.
- VLAN 146 contains servers running FTP, WWW, SMTP and DNS applications.
- Configure R4 to allow access to those applications from behind R5.
- Allow DNS zone file transfers in addition to DNS access.
- Ensure you only permit active FTP responses from servers in VLAN 146.
- Permit the exchange of RIP routing updates.
- Permit IOS “traceroute” and “ping” commands to function across R4.
- Ensure that the Path MTU discovery procedure works successfully across your firewall.
- Deny all ICMP “unreachable” messages with your configuration.
- Deny all other traffic.

11.6 Traffic Filtering with Reflexive Access-Lists

- Configure R5 to permit TCP, UDP, and ICMP packets sourced from behind the VLAN 58 interface.
- Do not create explicit access-list entries to permit returning TCP, UDP, and ICMP traffic.
- Ensure that Telnet and ICMP traffic originated from the router are also permitted by the access-list, but do not create any entries in the interface access-lists.
- Ensure RIP routing traffic is also permitted by your configuration.

11.7 Filtering Fragmented Packets

- Ensure R3 prevents fragmented packets from reaching its local HTTP server.
- Only configure the Frame-Relay interface for this task.

11.8 Access Control with Dynamic Access-Lists

- R6 should restrict access to the outside web servers for the users located on VLAN 67.
- Prior to being allowed to connect to a WWW server, a user should log in to the router using the name “ENABLE” and password of “CISCO”.
- The above procedure should only create an access-list entry for the IP address of the authenticated user.
- Alternatively, any user may connect to port 7001 of R6 and enter the password of “CISCO”.
- This procedure should enable any user on VLAN 67 and behind this VLAN to access the WWW servers.
- Close inactive connections after 5 minutes and do not allow sessions of more than 15 minutes in length.
- A user should be able to extend the maximum duration by logging into R6 repeatedly.

11.9 Traffic Filtering with Time-Based Access-Lists

- Restrict the users behind R2 from accessing BB1 on weekends.
- Additionally, disallow access to BB1 from 6pm to 9am on the remaining days.

11.10 Traffic Filtering with Policy-Based Routing

- Configure R6 to drop ICMP packets of 100 bytes in size entering the VLAN 146 interface.
- Ensure R6 only drops packets leaving via the VLAN 67 interface.
- The router should not send ICMP unreachable notifications about dropped packets.

11.11 Preventing Spoofing with uRPF

- R4 connects to your ISP at the border of your network.
- Ensure that R4 does not accept packets with IP addresses of the internal subnets on its connection to the ISP.
- There is another connection to the same ISP in the network, so account for possible asymmetric routing issues on R4.
- At the same time, R4 should only accept packets from legitimate subnets of 150.X.0.0/16 and 155.X.0.0/16 learned via IGP on its internal connections.
- Log all packets with spoofed sources.

11.12 Using NBAR for Content-Based Filtering

- Configure R6 to drop any potentially dangerous HTTP downloads.
- The dangerous files have extensions ".exe", ".com", or ".bin".
- Use a single pattern to match all the filenames at once.

11.13 TCP Intercept

- Configure R6 to protect the WWW servers on VLAN 146 from a SYN flood attack.
- Intercept connections as they go to the servers and close connections after 30 seconds of inactivity.
- Allow for a maximum of 100 semi-established connections and drop their number down to 80 when the threshold is crossed.
- The router should start dropping half-open connections when they cross the rate threshold of one connection per second. The router should not stop until the average rate is one per two minutes.
- The router should not account for connection age when dropping them.

11.14 TCP Intercept Watch Mode

- Modify your TCP intercept configuration in order to reset any connection that does not reach the established state in 20 seconds.

11.15 Packet Logging with Access-Lists

- Configure R6 to log all ICMP packets entering its VLAN 146 interface.
- Aggregate logging messages so that a log entry is generated after five access-list entry hits.
- Reduce the burden on the router CPU by process switching only one packet per second.
- Ensure the logged messages allow you to track the MAC address of the host sending the packet.

11.16 Stateful Filtering with CBAC

- Configure R4 as a stateful firewall. R4's VLAN 146 interface is the protected network and the Serial interface between R4 and R5 is the connection to outside networks.
- Ensure that TFTP and FTP protocols function across the firewall.
- Collect connection statistics for HTTP transfers.
- Allow pinging across the firewall but do not add any special ACL entries to accomplish this.
- Allow UDP sessions across the firewall with a 30 second inactivity timeout.
- Ensure that inspection entries for DNS requests expire in 10 seconds but do not inspect the DNS protocol for this.
- Disable CBAC alerts for all protocols with the exception of FTP.
- Do not allow any other TCP-based protocols except for those mentioned above.
- In the future some users are going to use the Cisco ezVPN client from inside the firewall. Ensure you do not need to add any ACL entries in order to allow ezVPN connections across the firewall.

11.17 Advanced CBAC Features

- Configure R4 so that you do not have to create manual openings for router-generated UDP traffic in the inbound access-lists.
- You expect no more than 5000 concurrent sessions in the busiest hours. Optimize CBAC performance to handle this configured maximum number of sessions.
- Only allow Java applets downloaded from the WWW servers in the subnet 150.X.5.0/24.
- Someone runs an FTP server listening on port 80. The IP address of the server is 150.X.5.25. Ensure that CBAC correctly inspects FTP sessions to this host.
- Ensure that Java filtering applies to connections through an outside HTTP proxy using port 8080.

11.18 CBAC TCP/UDP Intercept Feature

- Configure R6 to protect the servers on VLAN 146 subnet from TCP/UDP based DoS attacks per the requirements below.
- Allow for a maximum of 100 half-open connections and configure the firewall to keep dropping the sessions until their number reaches 80.
- Limit the new connections rate to 30 per minute and clamp it down to 15 connections per second once the threshold has been crossed.
- Limit the maximum number of half-open TCP connections per server to 10. Block any new connections for 60 seconds when the number exceeds this limit.

11.19 VLAN Filters for IP Traffic

- Restrict traffic on VLAN 146 so that only R1, R4, and R6 are allowed to communicate.
- Allow RIP routing updates as well as transit TCP connections on VLAN 146.
- Disallow any other traffic to cross VLAN 146.

11.20 VLAN Filters for Non-IP Traffic

- Allow only the following Layer 2 protocols across VLAN 146
  - STP BPDUs (consider both ISL and 802.1q trunks)
  - CDP, VTP, and UDLD
  - ARP
- Disallow any other Layer 2 packets.

11.21 Port Security

- Configure the respective switches to guard VLAN146 from MAC address flooding attacks originated at R1, R4, or R6.
- Limit the number of MAC addresses learned simultaneously on a single port to one.
- In case of a policy violation, apply the shutdown action on the port connected to R1 but recover the port after 3 minutes.
- For the port connected to R4, drop offending packets and generate log records of the violation.
- For the port connected to R6, simply drop the offending traffic.
- Age the learned secure entries after 10 minutes of inactivity.
- Retain the MAC addresses learned on the port connected to R4 in the switch configuration.

11.22 HSRP and Port-Security

- Reconfigure R3 and R4 temporarily for this task.
- Configure R1 and R3 to emulate an HSRP virtual router on VLAN 146 with the IP address 155.1.146.254.
- Configure the R4's Fa0/1 interface and the R6's VLAN 146 interface to emulate a virtual HSRP router on VLAN 146 with the IP address 155.1.146.253.
- Ensure your new configuration works with the port-security enabled ports.
- Do not use the **`switchport port-security maximum`** command on the port connected to R4.

11.23 DHCP Snooping

- Configure R4 as a DHCP server and R1 as a DHCP client on VLAN 146.
- Configure SW1 to prevent potential DHCP attacks in such a way that it dynamically accounts for future hosts added to the VLAN 146 segment.
- SW1 should store the binding database in flash with the filename dhcp-bindings.txt, and use a 15 second delay between changes.
- Limit the amount of DHCP messages SW1 can receive from R1 to 10 per second.

11.24 DHCP Snooping and the Information Option

- Configure R6 to obtain IP address information via DHCP from R4.
- Ensure that SW2 inserts Option 82 in DHCP requests received from R6.
- SW2 should be configured so that it uses the string “SWITCH2” for Option 82 Remote-ID and the string “ROUTER6” as the Circuit-ID for the port connected to R6.
- SW2 should accept information options in DHCP packets even on non-trusted ports.

11.25 Dynamic ARP Inspection

- Configure SW2 to prevent ARP poisoning attacks on VLAN 146.
- Do not trust any trunk ports to other switches, but ensure that the switch enforces ARP security for R4 and R1.
- Log all ARP packets permitted by the ARP access-list but limit their rate to four per ten seconds.
- Additionally, log all packets matched against the DHCP snooping database.
- Store, at maximum, 16 entries in the ARP logging buffer.
- Enable all additional sanity checks for ARP packets.

11.26 IP Source Guard

- Configure SW2 to prevent IP address spoofing on the port connected to R6.
- Ensure that your solution accounts for dynamic IP addresses obtained via DHCP.
- Enforce Layer 2 filtering for the MAC addresses corresponding to secured IP addresses at the same time.
- Do not shutdown ports in case of a security violation.

11.27 Using Catalyst Ingress Access-Lists

- Allow R1 to only send ARP and ICMP packets out of its VLAN 146 connection.

11.28 Controlling Terminal Line Access

- Only allow Telnet connections to R2 from the Loopback0 subnets.
- Authenticate remote users locally using the name “TELNET” and the password of “CISCO”.
- Only allow this user to connect to R1 from R2.
- Log any attempt to connect from R2 to any destination on port 80.

11.29 IOS Login Enhancements

- After 3 unsuccessful attempts to log into R3 in 30 seconds, block all further attempts for 40 seconds.
- Ensure that sessions sourced from the R5 Loopback0 are exempted from this restriction.
- Log every successful login attempt and every 3rd unsuccessful attempt.
- Enforce a delay of 2 seconds between successive login attempts.
- Create a local username “TEST” with the password of “TEST” for this task.

11.30 Role Based CLI

- Create roles in R4 named “SUPER”, “DEBUG”, “INTERFACE1” and “INTERFACE2”.
- The role “INTERFACE1” should have access to all IP configuration commands of interface Fa0/0 only. For “INTERFACE2” do the same with the exception that it applies to interface Fa0/1.
- The role “DEBUG” should be able to use any **debug** and **undebug** commands, and it should be able to inspect the running configuration.
- The last role name “SUPER” should be a sum of all the other roles.
- Use the password of “CISCO” to authenticate all roles.

11.31 IP Source Tracker

- You suspect that SW1 is under a distributed DoS attack from unidentified sources.
- Configure R6 to collect attack information for the VLAN 67 IP address of SW1.
- Export the statistics via syslog messages every 5 minutes.

11.32 Router IP Traffic Export

- Configure R1 to export the traffic sent and received on the Frame-Relay interface to the MAC address of R4.
- Configure a traffic filter that only allows TCP and ICMP packets to be monitored.

11.33 Controlling the ICMP Messages Rate

- In the future, you plan to apply an input traffic filter on R4's VLAN 146 interface.
- Configure this interface not to respond with an ICMP message when the filter drops a packet.
- The rate of these messages sent out of all other interfaces should not exceed 100 per second.
- In order to ensure that PMTUD works correctly, increase the rate of ICMP messages used by this feature to 1000 per second on R4.

11.34 Control Plane Policing

- Limit the aggregate rate of ARP traffic towards R1's route processor to 100 packets per second.
- The routing control traffic marked with an IP Precedence value of 6 should be limited to 50 packets per second.
- Drop all Telnet connections sourced from R5's Frame-Relay IP address.
- Limit the rate of outgoing ICMP messages to 10 per second.

Note

Erase the running configurations of all devices and load the *Security* initial configurations.

11.35 Control Plane Protection

- Enhance R3 protection by dropping packets going to all closed ports.
- Ensure this does not affect connections made on ports 3020 and 4040.
- To reduce the effect of broadcast storms, limit the rate of input non-IP packets to 100 per second
- Set the queue-limit for input HTTP packets to 100 packets and limit the packet rate to 10 per second.
- Ensure that all fragmented packets transiting the router are limited to 1 Mpps.

11.36 IOS ACL Selective IP Option Drop

- Configure R3 to drop any packets destined to the router with IP source route option (either loose or strict).
- Apply this protection to the link connected to SW1.

11.37 BGP Generic TTL Security Mechanism

- Configure eBGP session between R5 and R6 and make both routers accept the peer if it’s no more than one hop away.
- Use AS numbers 500 and 600 for R5 and R6 respectively.

11.38 Flexible Packet Matching

- Configure R1 to filter ICMP Echo packets with the string “AAA” in the payload. Look no deeper than 256 bytes in the packet.
- Ensure the filtering applies only to ICMP/IP packets received in Ethernet frames.

11.39 Zone Based Firewall

- Configure three security zones in R3: INSIDE, OUTSIDE and DMZ for the interfaces facing R1, R2 and SW1 respectively.
- Only allow R3 to be accessed using SSH and HTTPs from zones OUTSIDE and DMZ.
- Allow the INSIDE users to use the following protocols when accessing OUTSIDE: HTTP, FTP, ICMP, DNS, SSH, Telnet, AOL Instant Messenger.
- Account for the AOL Messenger using the non-standard port 80 and users connecting to HTTP proxies on ports 3128 and 8080.
- Allow the OUTSIDE users to access the servers in DMZ using the HTTP, FTP, DNS and TACACS+ protocols.
- The INSIDE users should have the same set of protocols allowed to zone DMZ with the addition of SSH and HTTPS.
- Preserve IP routing with your configuration and allow the users logged into the router to ping/telnet to OUTSIDE and DMZ zones.
- Limit the inside users to subnet 150.X.1.0/24 and the DMZ server to the subnet 155.X.37.0/24.

11.40 ZFW Rate Limiting

- In order to prevent DoS attacks, limit the OUTSIDE to DMZ traffic flow to 512Kbps.
- Enable half-open session limiting, so that no more than 2000 half-open sessions are allowed and no more than 100 new half-open sessions generated per minute are permitted.
- Stop dropping the half-open session states once the absolute amount of half-open connections falls below 1000 and the rate falls below 10.

11.41 ZFW Application Inspection

- Configure the firewall so that the internal users are not allowed to use insecure POP3/IMAP4 logins to the outside servers.
- Block image downloads from digg.com for internal users and log any attempts to use AOL messenger for non text-chat service (e.g. file transfer).
- Image files have filename extensions .jpg, .png, .gif
- Log all violations.

11.42- Classic IOS Transparent Firewall

- Configure R6 as a transparent firewall deployed on the subnet 10.0.0.0/24.
- Change the IP addresses for VLAN67 and VLAN146 interface for SW1, R1 and R4 to reflect this.
- VLAN67 users are allowed to originate HTTP, FTP, ICMP, DNS and SSH traffic across the firewall to VLAN146.
- VLAN146 users are only allowed initiating FTP (passive), HTTP and DNS connections to the serves on VLAN 67
- Do not permit any IPv6 traffic across the firewall.

11.43 ZFW-Based IOS Transparent Firewall

- Modify the previous scenario to use CPL for the firewall configuration.

11.44 IOS IPS

- Configure R6 to deny inline the attacker that attempts to use IP loose source route option.
- For testing purposes, enable the ICMP echo signature and configure the IPS to notify syslog.

Security Solutions

11.1 AAA Authentication Lists

- Configure R1 to use a TACACS+ server at the IP address 155.X.146.100 and encrypt communications using a key value of CISCO.
- Source TACACS+ packets from the Loopback0 interface.
- Configure the router so that access to the console line uses local authentication.
- Ensure the VTY lines authenticate users with TACACS+ and then fall back to line authentication.
- Privilege mode authentication should first attempt TACACS+ and then fall back to the local password.
- Create a user named “ADMIN” with a password of “CISCO” in the local database for these configurations.
- Customize the prompts for AAA user authentication and change the default banner message.

Configuration

R1:

```
!
! Enable password and a local user
!
enable secret cisco
username ADMIN password CISCO
!
! Init AAA and configure AAA lists for Console/VTY logins
! Enable the use of TACACS+ for enable authentication but
! provide a fallback to local enable password
!
aaa new-model
aaa authentication login CONSOLE local
aaa authentication login VTY group tacacs+ line
aaa authentication enable default group tacacs+ enable
!
! Customize prompts
!
aaa authentication password-prompt "Please Enter Your Password:"
aaa authentication username-prompt "Please Enter Your ID:"
!
! Add a new authentication banner
!
aaa authentication banner #
This system requires you to identify yourself.
#
```

```
aaa authentication fail-message #
Authentication Failed, Sorry.
#
!
ip tacacs source-interface loopback 0
!
tacacs-server host 155.1.146.100
tacacs-server directed-request
tacacs-server key CISCO
!
line con 0
 login authentication CONSOLE
!
line vty 0 4
 login authentication VTY
 password cisco
```

Verification

Note

Cisco routers and switches may now use a “new” authentication model. The idea of the new model is to apply configurable lists of methods to authenticate, authorize, and/or perform accounting for certain processes. The processes that we are typically most interested in are:

1) Login process – the procedure to obtain access to the router, usually requiring you to present your identity.

2) Privileged mode access - in order to access a certain level of “enable mode” privileges, the system requires user authentication. You can configure local enable passwords for each level of access. You can also configure a named list of methods using the command **`aaa authentication <process> {default|<NAME>} <List of Methods>`** You may apply this list to a certain subsystem, e.g. to terminal or console lines. Note that a “default” list applies to all subsystems that have no explicit AAA list applied to them.

As for the list of possible methods, we should be most concerned with the following:

1) “Local” method – use the local user database with their passwords. You populate the database using the **`username`** command.

2) “Local-case” – the same as the local method, but makes passwords case-sensitive.

3) “Line” method – use the password configured on the line used to access the router. This includes VTY lines as well.

4) “Enable” method – use the globally configured list of enable passwords associated with their levels.

5) “Group” TACACS+ or RADIUS – uses the remote AAA servers group configured globally in the router.

6) “None” – do not attempt to validate user identity, just allow access.

Note an important property of the AAA list. The router tries all methods in sequence, switching to the next one if the previous method fails or returns an error. Here “Fail” means authentication method failure, e.g. non-responsive server, not the failure of the user account to be authenticated. This means that list allows you to provide authentication redundancy. For example, consider a list which uses TACACS+ authentication first, and then switches to method “None”. In the case that your TACACS+ server does not respond, the router tries the next method and allows access. However, if the AAA server returns an authentication failure message, the router terminates the list search process, and denies access to the individual.

In this task configuration, you should verify the authentication process by trying to log in to the console line first:

```
Rack1R1 con0 is now available





Press RETURN to get started.



This system requires you to identify yourself.

Please Enter Your ID:ADMIN
Please Enter Your Password:CISCO

Rack1R1#
```

Note that the privilege level of the console line has been set to 15 by the initial configurations and thus the user logs directly to privileged mode.

Now enable AAA debugging and try logging in via a VTY line:

```
Rack1R1#debug aaa authentication
AAA Authentication debugging is on

Rack1R1#telnet 150.1.1.1
Trying 150.1.1.1 ... Open

AAA/BIND(0000000D): Bind i/f
AAA/AUTHEN/LOGIN (0000000D): Pick method list 'VTY'
This system requires you to identify yourself.

Please Enter Your Password:cisco
AAA/AUTHEN/LINE(0000000D): GET_PASSWORD
```

The router attempts to contact the TACACS+ server, fails and uses the second method, which is the line password.

```
AAA/AUTHEN/LINE(0000000D): PASS
AAA/AUTHOR (0000000D): Method=None for method list id=00000000. Skip
author

Rack1R1>enable
```

Now try entering the wrong password for privileged mode authentication. Note that the system tries the TACACS+ method first, but it returns the “ERROR” message. Thus, the system tries using the enable password for authentication, but the password entered does not match. Note the “Status=FAIL” response when you enter the wrong password.

```
AAA: parse name=tty66 idb type=-1 tty=-1
AAA: name=tty66 flags=0x11 type=5 shelf=0 slot=0 adapter=0 port=66
channel=0
AAA/MEMORY: create_user (0x85646E6C) user='NULL' ruser='NULL' ds0=0
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 initial_task_id='0', vrf= (id=0)
AAA/AUTHEN/START (2250436155): port='tty66' list='' action=LOGIN
service=ENABLE
AAA/AUTHEN/START (2250436155): using "default" list
AAA/AUTHEN/START (2250436155): Method=tacacs+ (tacacs+)
TAC+: send AUTHEN/START packet ver=192 id=-2044531141
AAA/AUTHEN(2250436155): Status=ERROR

AAA/AUTHEN/START (2250436155): Method=ENABLE
AAA/AUTHEN(2250436155): Status=GETPASS
AAA/AUTHEN/CONT (2250436155): continue_login (user='(undef)')

Please Enter Your Password: 12345

AAA/AUTHEN(2250436155): Status=GETPASS
AAA/AUTHEN/CONT (2250436155): Method=ENABLE
```

```
AAA/AUTHEN(2250436155): password incorrect
AAA/AUTHEN(2250436155): Status=FAIL
AAA/MEMORY: free_user (0x85646E6C) user='NULL' ruser='NULL'
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 vrf= (id=0)en

% Access denied
```

Try authenticating again, entering the correct password this time. Note that the system tries TACACS+ method again and then switches to the “enable” method. This time passwords match and the reply is “PASS”.

```
Rack1R1>enable

AAA: parse name=tty66 idb type=-1 tty=-1
AAA: name=tty66 flags=0x11 type=5 shelf=0 slot=0 adapter=0 port=66
channel=0
AAA/MEMORY: create_user (0x848DC170) user='NULL' ruser='NULL' ds0=0
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 initial_task_id='0', vrf= (id=0)
AAA/AUTHEN/START (649338587): port='tty66' list='' action=LOGIN
service=ENABLE
AAA/AUTHEN/START (649338587): using "default" list
AAA/AUTHEN/START (649338587): Method=tacacs+ (tacacs+)
TAC+: send AUTHEN/START packet ver=192 id=649338587
AAA/AUTHEN(649338587): Status=ERROR

Please Enter Your Password:cisco

AAA/AUTHEN/START (649338587): Method=ENABLE
AAA/AUTHEN(649338587): Status=GETPASS
AAA/AUTHEN/CONT (649338587): continue_login (user='(undef)')
AAA/AUTHEN(649338587): Status=GETPASS
AAA/AUTHEN/CONT (649338587): Method=ENABLE
AAA/AUTHEN(649338587): Status=PASS
AAA/MEMORY: free_user (0x848DC170) user='NULL' ruser='NULL'
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 vrf= (id=0)

Rack1R1#
```

In the debugging output above, note that username is “null” or “undef” when you log in via the VTY line. This is because we used the “line” password for authentication and never supplied a username.

11.2 AAA Exec Authorization

- Authorize exec privilege levels using the remote TACACS+ server.
- Ensure that console line logins are also authorized. The console line should authorize users with the TACACS+ method first and then the local database configuration.
- User “ADMIN” should be granted a privilege level of 7 if authenticated.
- Use the VTY line settings to authorize any authenticated users in the event the TACACS+ server fails.
- Ensure that if the TACACS+ server is unavailable, authenticated users logging in via VTY lines are placed in privilege level 15.

Configuration

```
R1:
!
! Enable EXEC authorization on the console line
!
aaa authorization console
aaa authorization exec default none
aaa authorization exec CONSOLE group tacacs+ local
aaa authorization exec VTY group tacacs+ if-authenticated
!
username ADMIN privilege 7 password 0 CISCO
!
line con 0
 authorization exec CONSOLE
!
line vty 0 4
 privilege level 15
 password cisco
 authorization exec VTY
 login authentication VTY
```

Verification

Note

Authorization is a procedure for granting certain rights to a process, or granting a permission to perform a certain action. The authorization procedure is only possible for authenticated entities. The identity of a subject is used to look up the policy and determine the permissions. This is why authentication always precedes authorization. In some cases, it is possible to grant some rights to unidentified subjects.

The goal of exec authorization is assigning a privilege level (0-15) to a logged in user. You configure an exec authorization list using the command:

```
aaa authorization exec {default|<NAME>} <Method List>
```

As with authentication, you can define a default list (which is used system wide) or apply a specific list per terminal line. Generally, there are three methods to obtain authorization information:

1) Consult a remote AAA server and download the user attributes. TACACS+ performs this procedure as a separate operation, but RADIUS has no explicit authorization state, and returns authorization information in authentication replies. Here is an example of using TACACS+ as the source of the required information:

```
aaa authorization exec default group tacacs+
```

2) Consult the local username database, looking for the privilege level assigned to the authenticated user:

```
aaa authorization exec default local
```

3) Use default settings, for example, the default privilege level assigned to the terminal line, if the authorization configuration permits. This is commonly used when you disable authorization (method “none”) or authorize settings for any authenticated users (method “if-authenticated”). Note the difference between the method “none” and “if-authenticated” from the following example:

Scenario 1:

```
aaa authentication login default tacacs+ none
aaa authorization exec default none
!
line console 0
 privilege level 15
```

Scenario 2:

```
aaa authentication login default tacacs+ none
aaa authorization exec default if-authenticated
!
line console 0
 privilege level 15
```

In the first case, if the TACACS+ server is not available, the router will allow incoming console connections without authentication. Since there is not exec authorization, the user will be granted the exec shell with privilege 15. In the second case, if the TACACS+ server is not available, the system grants access without authentication but fails authorization of exec shell.

Thus, the difference between “none” and “if-authenticated” authorization cases is that the former always applies the desired authorization parameters without any verification. The latter requires the user to be authenticated, but does not consult the user database to check authorization attributes.

By default, exec authorization is set to “none”, so you may need to change it to accomplish your needs. Also, note that IOS routers by default do not authorize exec sessions on the console line. On the contrary, Catalyst IOS always authorizes the exec shell, even on the console line. Therefore, if you disable console authentication in the Catalyst switch, make sure you never apply a AAA authorization list to the console (explicitly or using the default settings). You may enable console exec authorization in IOS routers using the command **`aaa authorization console`**.

For our sample task, first verify the authentication process by trying to log in to the console line.

```
Rack1R1 con0 is now available





Press RETURN to get started.



This system requires you to identify yourself.

Please Enter Your ID:ADMIN
Please Enter Your Password:CISCO

Rack1R1#show privilege
Current privilege level is 7
```

Note that the privilege level assigned to the user is seven, as specified by the local configuration (username command). The router first tries to use TACACS+ for exec authorization and falls back to the local configuration since the TACACS+ server is not reachable.

Now enable AAA debugging and try logging in via a VTY line.

```
Rack1R1#debug aaa authentication
AAA Authentication debugging is on

Rack1R1#telnet 150.1.1.1
Trying 150.1.1.1 ... Open

AAA/BIND(0000000D): Bind i/f
AAA/AUTHEN/LOGIN (0000000D): Pick method list 'VTY'
This system requires you to identify yourself.

Please Enter Your Password:cisco
AAA/AUTHEN/LINE(0000000D): GET_PASSWORD
```

The router attempts to contact the TACACS+ server, fails, and uses the second method, the line password.

```
AAA/AUTHEN/LINE(0000000D): PASS
AAA/AUTHOR (0000000D): Method=None for method list id=00000000. Skip
author
```

```
Rack1R1>enable
```

Now try entering the wrong password for enable authentication. Note that the system tries the TACACS+ method first, but it returns the “ERROR” message. Thus, the system tries using the enable password for authentication, but the password entered does not match. Note the “Status=FAIL” response when you enter the wrong password.

```
AAA: parse name=tty66 idb type=-1 tty=-1
AAA: name=tty66 flags=0x11 type=5 shelf=0 slot=0 adapter=0 port=66
channel=0
AAA/MEMORY: create_user (0x85646E6C) user='NULL' ruser='NULL' ds0=0
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 initial_task_id='0', vrf= (id=0)
AAA/AUTHEN/START (2250436155): port='tty66' list='' action=LOGIN
service=ENABLE
AAA/AUTHEN/START (2250436155): using "default" list
AAA/AUTHEN/START (2250436155): Method=tacacs+ (tacacs+)
TAC+: send AUTHEN/START packet ver=192 id=-2044531141
AAA/AUTHEN(2250436155): Status=ERROR

AAA/AUTHEN/START (2250436155): Method=ENABLE
AAA/AUTHEN(2250436155): Status=GETPASS
AAA/AUTHEN/CONT (2250436155): continue_login (user='(undef)')

Please Enter Your Password: 12345

AAA/AUTHEN(2250436155): Status=GETPASS
AAA/AUTHEN/CONT (2250436155): Method=ENABLE
AAA/AUTHEN(2250436155): password incorrect
AAA/AUTHEN(2250436155): Status=FAIL
AAA/MEMORY: free_user (0x85646E6C) user='NULL' ruser='NULL'
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 vrf= (id=0)en

% Access denied
```

Try authenticating again, entering the correct password this time. Note that the system tries the TACACS+ method again and then switches to the “enable” method. This time passwords match and the reply is “PASS”.

```
Rack1R1>enable

AAA: parse name=tty66 idb type=-1 tty=-1
AAA: name=tty66 flags=0x11 type=5 shelf=0 slot=0 adapter=0 port=66
channel=0
AAA/MEMORY: create_user (0x848DC170) user='NULL' ruser='NULL' ds0=0
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 initial_task_id='0', vrf= (id=0)
AAA/AUTHEN/START (649338587): port='tty66' list='' action=LOGIN
service=ENABLE
AAA/AUTHEN/START (649338587): using "default" list
AAA/AUTHEN/START (649338587): Method=tacacs+ (tacacs+)
TAC+: send AUTHEN/START packet ver=192 id=649338587
AAA/AUTHEN(649338587): Status=ERROR

Please Enter Your Password:cisco

AAA/AUTHEN/START (649338587): Method=ENABLE
AAA/AUTHEN(649338587): Status=GETPASS
AAA/AUTHEN/CONT (649338587): continue_login (user='(undef)')
AAA/AUTHEN(649338587): Status=GETPASS
AAA/AUTHEN/CONT (649338587): Method=ENABLE
AAA/AUTHEN(649338587): Status=PASS
AAA/MEMORY: free_user (0x848DC170) user='NULL' ruser='NULL'
port='tty66' rem_addr='150.1.1.1' authen_type=ASCII service=ENABLE
priv=15 vrf= (id=0)

Rack1R1#show privilege
Current privilege level is 15
```

In the debugging output above note that the username is always “null” or “undef” when you log in via the VTY line. This is because we used the “line” password for authentication and never supplied a username.

11.3 AAA Local Command Authorization

- Ensure that the user “ADMIN” can use RIP debugging commands and can disable any currently active debugging using a single command.
- The same user should be able to configure any interface IP settings and administratively enable or disable any of these interfaces.
- Ensure the user can see their permitted commands in their running configuration.

Configuration

```
R1:
privilege exec level 7 configure terminal
privilege exec level 7 undebug all
privilege exec level 7 show running-config
privilege exec level 7 debug ip rip
!
privilege configure level 7 interface
!
privilege interface level 7 shutdown
privilege interface level 7 no shutdown
privilege interface all level 7 ip
```

Verification

Note

IOS allows configuring command authorization by using the local configuration database. Command authorization permits specific commands to groups of users. IOS also supports remote command authorization with the TACACS+ protocol, but this is out of the scope of the CCIE R&S lab exam.

Local command authorization uses the concept of privilege levels. There are sixteen levels supported, 0 to 15. Every next level supports the commands found in all previous levels, e.g. privilege 5 includes levels 0-5, and privilege 15 includes levels 0-15. By default, IOS has three privilege levels pre-configured:

Level 0 - just a few basic commands, such as **`enable`**, **`login`**, and **`exit`**

Level 1 - the default exec user level; has some show commands available, but no configuration commands

Level 15 - the maximum privilege level, also known as privileged mode or enable mode; includes all the commands available in IOS

In order to create custom command sets, you can perform one of the following:

1) Assign some level 15 commands to level 1, effectively making them available to all users that may log in to the router (if they use the default privilege level settings). You may want to use this option if you need to allow all users the use of some special features, e.g. using certain debug commands.

2) Re-assign some commands from level 1 to a higher level, thus disallowing all unprivileged users the use of this command. For example, you may want to disallow the use of the **`show ip access-list`** command for all default privilege users.

3) Assign some level 15 commands to a new custom level, e.g. level 7. By doing this, you still make commands available to level 15, but do not allow any user with the default privilege to use them. After that, you can assign the custom privilege level to a specific user, allowing the use of some privileged commands to this particular user only.

To understand the command authorization process, you need to remember that at all times the IOS exec shell works in one of the interpreter modes. The two most well known modes are "exec mode" and "global configuration mode". The interpreter's mode is shown by default through the router's prompt, such as **`router#`** (exec) or **`router(config)#`** (global configuration). In addition to that, the shell contains many other interpreter modes, such as "interface configuration", "vpdn configuration", "ip extended ACL", "map-class", and so on. Each mode has its own subset of commands, which are only visible in the particular mode.

In order to see how IOS performs authorization, you need to understand the generic command syntax as well. Each command generally has the following structure:

```
command sub-command [arguments] [argument-values] [options]
```

Here *command* is the first sub-string you send to the interpreter, for example, "**`ip`**" in the "**`ip address`**" command under interface configuration mode. The *sub-command* field may be present in some commands, e.g. "**`ip proxy-arp`**", "**`compress stac`**" etc. The *arguments* cover all named parameters that may take values and that are mandatory. For example, in the "**`ip address`**" command, the "**`address`**" field is an argument and it may take a value such as "**`1.2.3.4`**". The *options* may cover various command attributes that are not mandatory.

From the local command authorization standpoint, you can only match the "non-variable" or "mandatory" fields such as "command", "sub-command", and "arguments". The system will automatically allow any argument values and options if the command that user enters match the configured pattern.

The syntax to re-assign a particular command is as follows:

`privilege <mode> level <level> <command>`

This command instructs the shell to assign the command matching the string "command" to the level specified by the "level" argument. Note that the match is performed against all mandatory parts of the command that a user enters in a particular exec mode. For example, if you assigned just the command **`snmp-server`** to level 7 but not the command **`snmp-server host`**, then a user will not be able to configure the SNMP traps destination, since "host" is a mandatory (non-optional) part of the command. The following features help you in configuring local command authorization:

1) When you enter commands as a shortcut, such as **`privilege exec level 7 conf t,`** the shell expands it to the full names, for example to **`privilege exec level 7 configure terminal`**

2) When you allow a compound command to a particular level, e.g. **`privilege interface level 7 ip address`** then the shell automatically adds another line allowing the first "sub-component" of the compound command, e.g. adds a **`privilege interface level 7 ip`** command.

3) You can use the special keyword "all" to allow any sub-command or argument behind a specific keyword. For example, the command **`privilege interface all level 7 ip`** allows all "ip" subcommands in the interface configuration mode.

The last thing to keep in mind is how the local command authorization interoperates with the **`show running-config`**. A special feature is that a user may only see the command they are authorized to use. For example, if a user is authorized to use the **`interface`** command and all "ip" subcommands, then the **`show running-config`** command will only display those specific commands.

Verification for this task consists of checking the commands that you are allowed to use:

```
Rack1R1 con0 is now available

Press RETURN to get started.

This system requires you to identify yourself.

Please Enter Your ID:ADMIN
Please Enter Your Password:CISCO

Rack1R1#show privilege
Current privilege level is 7

Rack1R1#?
Exec commands:
  access-enable    Create a temporary Access-List entry
  access-profile   Apply user-profile to interface
  call             Voice call
  clear            Reset functions
  configure        Enter configuration mode
  <snip>

Rack1R1#debug ?
  all           Enable all debugging
  call          Call Information
  call-mgmt     Call Management
  ces-conn      Connection Manager CES Client Info
  conn          Connection Manager information
  dspapi        Generic DSP API
  flow-sampler  Debug flow sampler
  hpi           HPI (54x) DSP messages
  ip            IP information
  ncia          Native Client Interface Architecture (NCIA) events
  network       Network related debugging
  rscmon        Resource Monitor

Rack1R1#debug ip rip ?
  <cr>

Rack1R1#undebug all
All possible debugging has been turned off

Rack1R1#conf ?
  terminal  Configure from the terminal
  <cr>
```

```
Rack1R1#conf terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R1(config)#?
Configure commands:
  atm          Enable ATM SLM Statistics
  call         Configure Call parameters
  default      Set a command to its defaults
  end          Exit from configure mode
  exit         Exit from configure mode
  help         Description of the interactive help system
  interface    Select an interface to configure
  no           Negate a command or set its defaults
  oer          Optimized Exit Routing configuration submodes

Rack1R1(config)#interface fastEthernet 0/0
Rack1R1(config-if)#?
Interface configuration commands:
  default   Set a command to its defaults
  exit      Exit from interface configuration mode
  help      Description of the interactive help system
  ip        Interface Internet Protocol config commands
  no        Negate a command or set its defaults
  shutdown  Shutdown the selected interface

Rack1R1(config-if)#ip ?
Interface IP configuration subcommands:
  access-group        Specify access control for packets
  accounting          Enable IP accounting on this interface
  address             Set the IP address of an interface
  admission           Apply Network Admission Control
  auth-proxy          Apply authentication proxy
  authentication      authentication subcommands
  bandwidth-percent   Set EIGRP bandwidth limit
  bgp                 BGP interface commands
  <snip>
```

Now inspect the running configuration contents:

```
Rack1R1#show run
Building configuration...

Current configuration : 527 bytes
!
! Last configuration change at 12:31:51 UTC Wed Sep 24 2008 by ADMIN
! NVRAM config last updated at 05:01:32 UTC Wed Sep 24 2008
!
boot-start-marker
boot-end-marker
!
!
!
!
!
interface Loopback0
 ip address 150.1.1.1 255.255.255.0
 ip rip advertise 10
!
interface FastEthernet0/0
 ip address 155.1.146.1 255.255.255.0
 ip rip advertise 10
!
interface Serial0/0
 ip address 155.1.0.1 255.255.255.0
 ip rip advertise 10
 ip split-horizon
!
interface Serial0/1
 ip address 155.1.13.1 255.255.255.0
 ip rip advertise 10
```

Note that you only see the **`interface`** and **`ip`** commands in the running configuration.

11.4 Traffic Filtering using Standard Access-Lists

- Using the minimum amount of ACL lines, allow access to R1 from the Loopback0 subnets of R2 through R6 that have an even 3rd octet.
- Deny access from any other “150.X.0.0/16” subnets.
- Any hosts from the “155.X.0.0/16” subnets should still be able to reach R1.
- Apply the access-list inbound on all interfaces of R1.

Configuration

```
R1:
access-list 1 permit 150.1.0.0 0.0.6.255
access-list 1 permit 155.1.0.0 0.0.255.255
!
interface FastEthernet 0/0
 ip access-group 1 in
!
interface Serial 0/1
 ip access-group 1 in
!
interface Serial 0/0
 ip access-group 1 in
```

Verification

Note

Standard ACLs only allow you to match source IP addresses based on “base” IP address and wildcard mask. Because of that “aggregate” behavior, standard ACLs are commonly configured at network nodes close to the “protected” object. One very common task is finding a required “base” IP address and wildcard mask pair based on set of requirements. Let us look at three different examples to illustrate how they could be solved.

Example 1 (the task):
Configure the IP/Wildcard Mask pair to match IP addresses in subnets 155.1.Y.0/24 where Y is an even number and is greater than or equal to 1 and less than or equal to 6.

Note that the 4th octet of the subnets may take any value from 0 to 255, so we just set the corresponding part of the wildcard mask to 255 (all 1’s).

Step 1:
We start by finding the fixed part of the subnets range. Since Y changes from 1 to 6, only the last 3 bits of the 3rd octet may vary. Thus, the topmost 6 bits of Y are fixed at zero. For the fixed part of the base IP address, we set wildcard bits to zeroes (meaning it does not change). Now write down all possible values that Y may take:

2=00000010
4=00000100
6=00000110

Step 2:
Find the part of the "bit-vector" that changes between different values. In this case, this part contains bits 1 and 2 (counting from the right and starting at zero). Create a mask vector, putting 1's in the positions where bit values in data vectors may change:

m=00000110=6

Essentially, this is the wildcard mask for the third octet.

Step 3:
Take into account the extra values covered by the mask. Note that the mask always cover 2^N values, where N is the number of 1's in the mask. In our case, we had three values, but the mask contains two non-zero bits resulting in four possible values. Effectively, our wildcard mask cover one extra bit vector:

0=00000000

Strictly speaking you may need a special ACL entry to deny the corresponding subnet. However, in our topology, there is no router with subnet 155.1.0.0/24, so you may ignore this extra value.

Step 4:
Compute the "base" 3rd octet value, taking the common part of all the bit vectors. Effectively you may do so by performing a logical bitwise AND over all values:

2 AND 4 AND 6 = 0.

Thus the resulting IP and Wildcard Mask pair is:

155.1.0.0 0.0.6.255

The resulting ACL may look like:

deny 155.1.0.0 0.0.0.255
permit 155.1.0.0.6.255

Note that the first line is NOT necessary in our case.

Example 2:
Configure the IP/Wildcard Mask pair to match IP addresses in subnets 155.1.Y.0/24 where Y is an odd number and is greater than or equal to 1 and less than or equal to 6.

Note that the 4th octet of the subnets may take any value from 0 to 255, so we just set the corresponding part of the wildcard mask to 255 (all 1's).

Step 1:
We start by finding the fixed part of the subnets range. Since Y changes from 1 to 6, only the last 3 bits of the 3rd octet may vary. Thus, the topmost 6 bits of Y are fixed to zero. For the fixed part of the base IP address, we set wildcard bits to zeroes (meaning it does not change). Now write down all possible values that Y may take:

1=00000001
3=00000011
5=00000101

Step 2:
Find the part of the "bit-vector" that changes between the different values. In this case, this part contains bits 1 and 2 (counting from the right and starting at zero). The zero bit is fixed to "1". Create a mask vector, putting 1's in the positions where bit values in data vectors may change:

m=00000110=6

Essentially, this is the wildcard mask for the third octet.

Step 3:
Take into account the extra values covered by the mask. Note that the mask always cover 2^N values where N is the number of 1's in the mask. In our case, we had three values, but the mask contains two non-zero bits resulting in four possible values. Effectively, our wildcard mask covers one extra bit vector:

7=00000111

Strictly speaking we need a special ACL entry to deny the corresponding subnet, since it is a part of our topology.

Step 4:
Compute the "base" 3rd octet value using the common part of all the bit vectors. Effectively, you may do so by performing a logical bitwise AND over all the values:

1 AND 3 AND 5 = 1.

Thus the resulting IP and wildcard mask pair is:

155.1.1.0 0.0.6.255

and the resulting ACL looks like:

deny 155.1.7.0 0.0.0.255
permit 155.1.1.0 0.0.6.255

In cases where octet values are widely separated, the resulting ACL may contain too many "deny" statements and it may be more effective to use just the entries matching specific values. In our case, the resulting ACL needs to also permit the 155.1.0.0/16 subnet as well.

Note that standard ACLs do not allow you to specify the protocol values and effectively permit any IP protocol for matching sources. The general use of standard ACLs is route filtering and prefix matching in routing protocol configuration. Note that standard ACLs may be named as well as numbered (1-99).

To verify your configuration, first make sure that the IP access list applies to the interfaces:

```
Rack1R1#sh ip interface serial 0/0 | inc acce
  Outgoing access list is not set
  Inbound  access list is 1
  IP access violation accounting is disabled

Rack1R1#sh ip interface serial 0/1 | inc acce
  Outgoing access list is not set
  Inbound  access list is 1
  IP access violation accounting is disabled

Rack1R1#sh ip interface FastEthernet 0/0 | inc acce
  Outgoing access list is not set
  Inbound  access list is 1
  IP access violation accounting is disabled
```

Verify that you can only reach R1 from the Loopbacks of R2, R4 and R6:

```
Rack1R2#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.2.2
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 84/86/88 ms

Rack1R3#ping 155.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.3.3
.....
Success rate is 0 percent (0/5)

Rack1R4#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.4.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Rack1R5#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.5.5
U.U.U
Success rate is 0 percent (0/5)

Rack1R6#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.6.6
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Rack1SW1#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.7.7
UU.UU
Success rate is 0 percent (0/5)

Rack1SW2#ping 150.1.1.1 source loopback 0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.8.8
.U.U.
Success rate is 0 percent (0/5)
```

11.5 Traffic Filtering using Extended Access-Lists

- Configure R4 to permit any TCP traffic destined for R5 and sourced from the Loopback0 subnets. Ensure R4 allows returning TCP packets for those connections.
- VLAN 146 contains servers running FTP, WWW, SMTP and DNS applications.
- Configure R4 to allow access to those applications from behind R5.
- Allow DNS zone file transfers in addition to DNS access.
- Ensure you only permit active FTP responses from servers in VLAN 146.
- Permit the exchange of RIP routing updates.
- Permit IOS “traceroute” and “ping” commands to function across R4.
- Ensure that the Path MTU discovery procedure works successfully across your firewall.
- Deny all ICMP “unreachable” messages with your configuration.
- Deny all other traffic.

Configuration

```
R4:
no ip access-list extended INBOUND
ip access-list extended INBOUND
remark ==
remark == Active FTP uses TCP ports 20 and 21
remark ==
permit tcp any 155.1.146.0 0.0.0.255 range 20 21
remark ==
remark == WWW and SMTP use port numbers 80 and 25
remark ==
permit tcp any 155.1.146.0 0.0.0.255 eq 80
permit tcp any 155.1.146.0 0.0.0.255 eq 25
remark ==
remark == DNS uses UDP port 53 and TCP port 53 for zone xfers
remark ==
permit udp any 155.1.146.0 0.0.0.255 eq 53
permit tcp any 155.1.146.0 0.0.0.255 eq 53
remark ==
remark == Established TCP sessions are traffic returning back
remark ==
permit tcp any 150.1.0.0 0.0.255.255 established
remark ==
remark == Inbound traceroute UDP probes
remark ==
permit udp any any range 33434 33474
remark ==
remark == Backscatter ICMP traffic for outbound traceroute UDP probes
remark ==
permit icmp any any port-unreachable
permit icmp any any time-exceeded
```

```
remark ==
remark == ICMP message used by PMTU discovery: Packet too big and DF
bit set
remark ==
permit icmp any any packet-too-big
remark ==
remark == Ping operation packets
remark ==
permit icmp any any echo
permit icmp any any echo-reply
remark ==
remark == RIP routing updates
remark ==
permit udp any any eq 520
remark ==
remark == Deny and log any other packets
remark ==
deny ip any any log
!
no ip access-list extended OUTBOUND
ip access-list extended OUTBOUND
remark ==
remark == Active FTP uses TCP ports 20 and 21
remark ==
permit tcp 155.1.146.0 0.0.0.255 range 20 21 any
remark ==
remark == WWW and SMTP use port numbers 80 and 25
remark ==
permit tcp 155.1.146.0 0.0.0.255 eq 80 any
permit tcp 155.1.146.0 0.0.0.255 eq 25 any
remark ==
remark == DNS uses UDP por 53 and TCP port 53 for zone xfers
remark ==
permit udp 155.1.146.0 0.0.0.255 eq 53 any
permit tcp 155.1.146.0 0.0.0.255 eq 53 any
remark ==
remark == TCP traffic from Loopback0 subnets
remark ==
permit tcp 150.1.0.0 0.0.255.255 any
remark ==
remark == Outbound traceroute UDP probes
remark ==
permit udp any any range 33434 33474
remark ==
remark == Backscatter ICMP traffic for inbound traceroute UDP probes
remark ==
permit icmp any any port-unreachable
permit icmp any any time-exceeded
remark ==
remark == ICMP message used by PMTU discovery: Packet too big and DF
bit set
remark ==
permit icmp any any packet-too-big
```

```
remark ==
remark == Ping operation packets
remark ==
permit icmp any any echo
permit icmp any any echo-reply
remark ==
remark == Deny and log any other packets
remark ==
deny ip any any log
!
interface Serial 0/0
 ip access-group INBOUND in
 ip access-group OUTBOUND out
!
interface Serial 0/1
 ip access-group INBOUND in
 ip access-group OUTBOUND out
```

Verification

Note

Extended ACLs allow the matching of source/destination IP addresses and source/destination ports. In addition to that, you can match ICMP message types, DSCP values, port ranges, TCP flags, and so on. Essentially, extended ACLs permit configuration of a very flexible stateless packet filter. Due to the stateless nature of the filtering, for each “outbound” rule you need a matching “inbound” rule, permitting the returning traffic. Thus, inbound and outbound extended access-lists usually mirror each other (e.g. port numbers swapped between source and destination IP addresses).

Extended ACLs allow limited “emulation” of stateful behavior for TCP connections. Specifically, by using the **`established`** keyword, you permit all TCP packets that have the “ACK” bit set. This does not include the initial “SYN-only” packet, and thus permits only the packets that are part of an established session.

In order to configure an extended ACL successfully, you need to know the details of application networking logic. At the very least, this includes the TCP/UDP port numbers used by the application. You may also need to know some additional details, for example, how active FTP differs from passive FTP, or how the traceroute utility works, etc. Most of the common ports can be found using the shell prompt and typing the question mark after **`permit tcp any any eq`**. You may also learn application ports by using the commands: **`show ip port-map`** or **`show ip nbar port-map`**.

Another special feature to keep in mind is functionality of outbound access-lists. The router's locally generated traffic is not subject to match against the outbound access-lists, only transit traffic matches outbound lists.

Next, we are going to discuss two applications commonly used in practice. The first one is FTP. Per the design, FTP uses two TCP connections, a control connection to send commands, and a data connection to transfer files. The control connection always uses a destination port 21. The data connection may work in one of two modes: active or passive.

1) In active mode, when a client issues a command requiring data transfer, the server instructs the client to listen on an ephemeral port (above 1024) and initiates a connection to the client sourced from port number 20. In this mode, the server connects to a client on a dynamic destination port.

2) In passive mode, when the client needs to transfer a file, the server opens a new ephemeral port (above 1024) and reports it to the client. The client then connects to the ephemeral port and the data transfer begins. In this mode, the client connects to the server on a dynamic destination port. Note that port 20 is *not used* in passive mode.

The next application to discuss is the traceroute utility. There are different variants of the traceroute utilized by various operating systems. IOS uses the so-called UNIX variant. In this variant, the client sends UDP probes with increasing TTL values starting at the TTL of 1. The client sends probes to incrementing UDP ports starting at 33434 for the first hop. The destination IP address in all packets is the target IP address.

1) If the next hop is not the ultimate destination, the receiving device checks the TTL field. If the TTL field expires, the device sends an ICMP time exceeded message sourced from its own IP address. The source node learns the hop number based on the UDP port number, encapsulated as part of the payload in the ICMP response message.

2) If the hop is the ultimate destination, it sends an ICMP port unreachable message for the port range selected specifically not to match any application. The source node learns the hop count based on the port number.

By default the traceroute utility only probes up to 30 hops, so the default UDP port range is 33434…33464.

Finally, the Path MTU Discovery (PMTUD) process relies on the ICMP message “packet too big”, also known as “fragmentation required but DF bit set”. Note that you can permit all ICMP unreachable messages using the **`unreachable`** keyword, or permit any ICMP message selectively.

During the creation of access-lists, use remarks extensively in real life configurations to make management and readability of function easier. In the lab exam, first try typing access-lists in notepad, it is easier to spot mistakes that way. In the lab, it’s also a good idea to add the additionally **`deny ip any any log`** in the end of your access list to catch any unnoticed traffic you may have not permitted.

In this verification, we will configure R6 to simulate some of the services, for example, HTTP and DNS:

```
R6:
ip http server
ip dns server
ip host TEST 150.1.6.6
```

Verify that you can only reach R1 from the Loopbacks of R2, R4 and R6:

```
Rack1R5#ping 155.1.146.6

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/17/20 ms

Rack1R5#traceroute 155.1.146.6

Type escape sequence to abort.
Tracing the route to 155.1.146.6

  1 155.1.45.4 12 msec
    155.1.0.4 16 msec
    155.1.45.4 8 msec
  2 155.1.146.6 20 msec *  16 msec

Rack1R5#telnet 155.1.146.6 80
Trying 155.1.146.6, 80 ... Open

Rack1R5#disc 1
Closing connection to 155.1.146.6 [confirm]
```

```
Rack1R5#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R5(config)#ip name-server 155.1.146.6
Rack1R5(config)#ip domain-lookup
Rack1R5(config)#^Z

Rack1R5#ping TEST

Translating "TEST"...domain server (155.1.146.6) [OK]

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.6.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 36/38/40 ms
```

Verify that you can originate TCP connections only from a Loopback0 interface, but not from the 155.1.0.0/16 subnets. After this verification, ensure that traceroute and ping work from behind R4.

```
Rack1R6#telnet 150.1.5.5
Trying 150.1.5.5 ...
% Destination unreachable; gateway or host down

Rack1R6#telnet 150.1.5.5 /source loopback 0
Trying 150.1.5.5 ... Open


User Access Verification

Password: cisco
Rack1R5>

Rack1R6#ping 150.1.5.5

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.5.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 36/38/40 ms

Rack1R6#traceroute 150.1.5.5

Type escape sequence to abort.
Tracing the route to 150.1.5.5

  1 155.1.146.4 0 msec 4 msec 0 msec
  2 155.1.0.5 16 msec *  16 msec
Rack1R6#
```

Be sure to check your access-list match counters:

```
**Rack1R4#sh ip access-lists**
Extended IP access list INBOUND
    10 permit tcp any 155.1.146.0 0.0.0.255 range ftp-data ftp
    20 permit tcp any 155.1.146.0 0.0.0.255 eq www (6 matches)
    30 permit tcp any 155.1.146.0 0.0.0.255 eq smtp
    40 permit udp any 155.1.146.0 0.0.0.255 eq domain (1 match)
    50 permit tcp any 155.1.146.0 0.0.0.255 eq domain
    60 permit tcp any 150.1.0.0 0.0.255.255 established (20 matches)
    70 permit udp any any range 33434 33474 (6 matches)
    80 permit icmp any any port-unreachable (2 matches)
    90 permit icmp any any time-exceeded
    100 permit icmp any any packet-too-big
    110 permit icmp any any echo (10 matches)
    120 permit icmp any any echo-reply (5 matches)
    130 permit udp any any eq rip (8094 matches)
    140 deny ip any any log
Extended IP access list OUTBOUND
    10 permit tcp 155.1.146.0 0.0.0.255 range ftp-data ftp any
    20 permit tcp 155.1.146.0 0.0.0.255 eq www any (3 matches)
    30 permit tcp 155.1.146.0 0.0.0.255 eq smtp any
    40 permit udp 155.1.146.0 0.0.0.255 eq domain any (1 match)
    50 permit tcp 155.1.146.0 0.0.0.255 eq domain any
    60 permit tcp 150.1.0.0 0.0.255.255 any (21 matches)
    70 permit udp any any range 33434 33474 (3 matches)
    80 permit icmp any any port-unreachable (2 matches)
    90 permit icmp any any time-exceeded
    100 permit icmp any any packet-too-big
    110 permit icmp any any echo (5 matches)
    120 permit icmp any any echo-reply (10 matches)
    130 deny ip any any log (1 match)
```

11.6 Traffic Filtering using Reflexive Access-Lists

- Configure R5 to permit TCP, UDP, and ICMP packets sourced from behind the VLAN 58 interface.
- Do not create explicit access-list entries to permit returning TCP, UDP, and ICMP traffic.
- Ensure that Telnet and ICMP traffic originated from the router are also permitted by the access-list, but do not create any entries in the interface access-lists.
- Ensure RIP routing traffic is also permitted by your configuration.

Configuration

```
R5:
no ip access-list extended OUTBOUND
ip access-list extended OUTBOUND
 permit tcp any any reflect MIRROR
 permit icmp any any reflect MIRROR
 permit udp any any reflect MIRROR
 deny ip any any log
!
no ip access-list extended INBOUND
ip access-list extended INBOUND
 permit udp any any eq 520
 evaluate MIRROR
!
interface Serial 0/0
 ip access-group INBOUND in
 ip access-group OUTBOUND out
!
interface Serial 0/1
 ip access-group INBOUND in
 ip access-group OUTBOUND out
!
! Local policy to divert telnet connections over the loopback
!
no ip access-list extended LOCAL_TRAFFIC
ip access-list extended LOCAL_TRAFFIC
 permit tcp any any eq 23
 permit icmp any any echo
!
route-map DIVERT_LOCAL
 match ip address LOCAL_TRAFFIC
 set ip next-hop 150.1.5.254
!
ip local policy route-map DIVERT_LOCAL
```

Verification

Note

Reflexive access-lists add “stateful” behavior to the IOS packet filtering features. The overall idea is simple: when a packet matches an ACL entry, instruct the router to create a “mirrored” entry in the access list applied in the opposite direction. For example, if a packet hits an inbound ACL, take the source/destination IP addresses and ports, swap them, and add dynamic entries to the outbound access list with the new parameters. This procedure automatically permits returning traffic for symmetric flows. It will not work for “non-standard” TCP applications, which are complicated protocols such as RCP, TFTP, and H323, which open dynamic channels with no way for a router to predict their behavior.

The reflexive access list implementation is as follows. When you configure a regular ACL entry for TCP, UDP, or ICMP traffic, you may add the command “**`reflect <ACL_NAME>`**”. When a packet matches this entry, the router adds the dynamic “mirrored” ACL entry to the ACL named <ACL_NAME>. Later, you can add the command “**`evaluate <ACL_NAME>`**” in the access-list applied in the opposite direction. This command instructs the router to evaluate the entries in the dynamically populated ACL to find a match for returning packet. There is no need to pre-create the dynamically populated ACL, the IOS creates it automatically.

Note that you may only reflect UDP, TCP and ICMP session traffic. All dynamic entries persist as long as the session is active. When a session becomes inactive, the router ages out the corresponding entry in the time specified by the global command:

```
ip reflexive-list timeout <TIMEOUT>
```

The default timeout value is 300 seconds.

Note the important feature that the dynamic access list is only populated when a packet hits an ACL entry. Also recall that locally generated router traffic does not match any outgoing ACLs. Thus, if you use reflexive ACLs, you may need to account for that feature and provide any of the following workarounds:

1) Create a static entry in the inbound access-list to permit the returning routing traffic. This works for protocols like RIP, OSPF, and EIGRP which send broadcast hello packets.

2) Use local policy routing to divert the local traffic across the loopback interface and make it re-enter the router, in effect, becoming transit traffic. This will cause a match for the outgoing ACLs and populate dynamic entries. This works for unicast session traffic, such as Telnet sessions off the router, ICMP packet flows, or BGP connections.

For verification, source ICMP packets off R5 and originate a telnet session. In both cases, look at the dynamically populated access-list MIRROR.

```
Rack1R5#ping 150.1.3.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.3.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 60/61/64 ms

Rack1R5#show ip access-list MIRROR
Reflexive IP access list MIRROR
     permit icmp host 150.1.3.3 host 155.1.0.5  (20 matches) (time left 296)

Rack1R5#telnet 150.1.3.3
Trying 150.1.3.3 ... Open


User Access Verification

Password:
Rack1R3>exit

[Connection to 150.1.3.3 closed by foreign host]

Rack1R5#show ip access-list MIRROR
Reflexive IP access list MIRROR
     permit tcp host 150.1.3.3 eq telnet host 155.1.0.5 eq 12864 (77 matches)
(time left 2)
     permit icmp host 150.1.3.3 host 155.1.0.5  (20 matches) (time left 282)
```

Next, check the total statistics accumulated by all access-lists. Note the matches for RIP updates.

```
Rack1R5#show ip access-list
Extended IP access list INBOUND
    10 permit udp any any eq rip (185 matches)
    20 evaluate MIRROR
Extended IP access list LOCAL_TRAFFIC
    10 permit tcp any any eq telnet (20 matches)
    20 permit icmp any any echo (5 matches)
Reflexive IP access list MIRROR
     permit icmp host 150.1.3.3 host 155.1.0.5  (20 matches) (time left 272)
Extended IP access list OUTBOUND
    10 permit tcp any any reflect MIRROR (20 matches)
    20 permit icmp any any reflect MIRROR (5 matches)
    30 permit udp any any reflect MIRROR
    40 deny ip any any log
```

11.7 Filtering Fragmented Packets

- Ensure R3 prevents fragmented packets from reaching its local HTTP server.
- Only configure the Frame-Relay interface for this task.

Configuration

```
R3:
no ip access-list extended NO_FRAGMENTS
ip access-list extended NO_FRAGMENTS
 permit udp any any eq rip
 deny ip any any fragments
 permit tcp any any eq 80
!
interface Serial 1/0
 ip access-group NO_FRAGMENTS in
```

Verification

Note

By default, the IP protocol allows packet fragmenting. This feature has been a long time known weakness, exploited by various types of attacks (e.g. ping of death). Overlapping fragments, fragments exceeding the assembly buffer, and fragments arriving out of order, are just a few examples of traffic that can severely degrade end system performance when an attacker sends them at high rates.

Additionally, fragmented packets are often used to bypass IDS or firewall systems. The attacker splits a packet in such a way that the firewall or IDS system is not able to extract information about something like the port numbers. Advanced firewalls and IDS systems support packet stream reassembly, but this procedure may degrade overall security system performance.

Due to these reasons, it is a good practice to avoid traffic fragmentation in your network and protect your servers against fragmented packets. One solution would be to configure matching MTU values and enable the PMTU Discovery process.

The IOS firewall is able to classify IP packets as one of the following:

1) Non-fragmented packets or initial fragments. These packets have a fragment offset of zero and commonly contain some upper-level protocol payload (TCP) to allow the extraction of information about application ports.

2) Non-initial fragments have a non-zero fragment offset and are the remaining parts of a fragmented packets. These packets do not have upper level protocol port information and the IOS firewall cannot match them against ACL entries configured with TCP/UDP port numbers. Instead of matching the port numbers, the firewall uses only Layer 3 information in an ACL entry (e.g. source/destination IP addresses) to match against the source/destination IP addresses in the packet.

This is why any non-initial fragment sourced from 1.1.1.1 to 2.2.2.2 matches the following ACL entry:

```
permit tcp host 1.1.1.1 host 2.2.2.2 eq 80
```

Even though it does not contain any port information, the firewall only matches the source and destination IP addresses for non-initial fragments.

You can match non-initial fragments using the **`fragments`** keyword in your ACL entry, for example:

```
deny ip any host 2.2.2.2 fragments
permit tcp host 1.1.1.1 host 2.2.2.2 eq 80
```

This configuration ensures that only non-fragmented packets and initial fragments may reach port 80 of the target system.

You can configure R5 so that outgoing TCP packets are fragmented. To accomplish this, ensure that PMTU Discovery is disabled on SW2 and the MTU is set to a minimum value on the Frame-Relay interface of R5. This will force R5 to fragment TCP/IP packets exceeding 68 bytes in size. You may need to remove the inbound ACL on R5 to permit returning traffic for the HTTP connection.

After this configuration, try connecting to R3 on port 80 from SW2 and download a file from the flash memory. The IP+TCP header size is 40 bytes and the payload size is most likely more than 28 bytes (the “GET” command and the URL) so the packets will surely exceed the IP MTU.

```
R3:
ip http server
ip http path flash:

Rack1SW2(config)#no ip tcp path-mtu-discovery
```

```
Rack1R5(config)#interface Serial 0/0
Rack1R5(config-if)#ip mtu 68
Rack1R5(config-if)#no ip access-group INBOUND in

Rack1SW2#copy http://admin:cisco@150.1.3.3/c3640-jk9o3s-mz.124-13a.bin
null:
%Error opening http://admin:cisco@150.1.3.3/c3640-jk9o3s-mz.124-13a.bin
(I/O error)
Rack1SW2#
```

The connection times out, and looking at the access-list statistics on R3 you can see exactly why. The reason is the packets are denied.

```
Rack1R3#show ip access-lists
Extended IP access list NO_FRAGMENTS
    10 permit udp any any eq rip (234 matches)
    20 deny ip any any fragments (27 matches)
    30 permit tcp any any eq www (24 matches)
```

Now change the MTU back to normal and try downloading again:

```
Rack1R5(config)#interface Serial 0/0
Rack1R5(config-if)#ip mtu 1500

Rack1SW2#copy http://admin:cisco@150.1.3.3/c3640-jk9o3s-mz.124-13a.bin
null:
Loading http://admin:cisco@150.1.3.3/c3640-jk9o3s-mz.124-13a.bin
```

Now R5 does not fragment the packets and the connection completes normally.

11.8 Filtering Packets with Dynamic Access-Lists

- R6 should restrict access to the outside web servers for the users located on VLAN 67.
- Prior to being allowed to connect to a WWW server, a user should log in to the router using the name “ENABLE” and password of “CISCO”.
- The above procedure should only create an access-list entry for the IP address of the authenticated user.
- Alternatively, any user may connect to port 7001 of R6 and enter the password of “CISCO”.
- This procedure should enable any user on VLAN 67 and behind this VLAN to access the WWW servers.
- Close inactive connections after 5 minutes and do not allow sessions of more than 15 minutes in length.
- A user should be able to extend the maximum duration by logging into R6 repeatedly.

Configuration

```
R6:
no aaa new-model
!
! Enable absolute timeout extension
!
access-list dynamic-extended
!
! Access-list with dynamic entries
!
no ip access-list extended 100
ip access-list extended 100
 permit tcp any any eq telnet
 permit tcp any any eq 7001
 permit udp any any eq 520
 dynamic ACCESS timeout 15 permit tcp any any eq 80
 deny ip any any log
!
! Apply the access-list inbound
!
interface FastEthernet 0/0.67
 ip access-group 100 in
!
! Note that password and autocommand are on separate lines
! This is because sometimes IOS thinks “autocommand” is a part
! of the password
!
username ENABLE password CISCO
username ENABLE autocommand access-enable host timeout 5
```

```
!
! First three lines authenticate users against local database
!
line vty 0 3
 login local
!
! Dedicate a special line for line-based authentication
! This line could be accessed on port 7001 (7000+rotary group number)
!
line vty 4
 rotary 1
 password CISCO
 login
 autocommand access-enable timeout 5
```

Verification

Note

Dynamic access-lists (also known as "lock and key") allow special types of entries, activated using the CLI command **`access-enable`**. Until this command triggers the entries, they are inactive and the IOS ignores them while inspecting an access-list. The **`access-enable`** command unlocks the dynamic entries, hence the name.

Note that once you execute **`access-enable`**, it activates all dynamic entries in all access-lists. Commonly, the activation command is bound to a specific user or router-line using the **`auto-command`** syntax. This results in a specific user logging in to a router and enabling the dynamic access-list entries.

Dynamic access-list entries time out after some time. There are two timeouts defined for dynamic ACLS:

1) The inactivity timeout; you specify this timeout using the command **`access-enable timeout <TIMEOUT>`**. This timeout only applies when no packets match the entry for the specified amount of time. The default is no idle timeout (only absolute timeout).

2) Absolute timeout; you specify this timeout under the access-list entry, for example

`access-list 100 dynamic <SOME_NAME> timeout <TIMEOUT> …`

This is the maximum amount of time that the entry may stay active. After this timeout expires, the user my log in and activate the command again. The default absolute timeout is infinite.

The common implementation of dynamic access-lists looks as follows:

1) Create an extended access-list (either named or numbered) and ensure it permits Telnet (or any other remote access method, for example SSH) to the local router. Populate the access-list with dynamic entries, but make sure it allows remote access to the router.

2) Apply the access-list inbound on the interface controlling user access. Dynamic entries only work in the inbound direction, and you cannot use them in outbound ACLs.

3) Create a local user with **`autocommand access-enable`** or apply this command to terminal lines (or a selected terminal line, using rotary groups). Tune the timeout values, if you need.

4) When a user wants to activate the set of access rules, he or she logs in via Telnet or SSH to the router and authenticates using name and password (or authenticates per the VTY line settings). After that, the **`auto-command`** triggers the dynamic ACL entries and terminates user connections immediately. Now the user may access outside resources per the activated dynamic rules.

By default, if you execute the command **`access-enable`** it activates the entry configured in the access-list. If you want to insert the source IP address of the user logging into the router, use the command **`access-enable host`**. This command replaces the source IP address specification (e.g. **any**) in the dynamic access-list entry with the IP address of the authenticated user.

Another special feature is absolute timeout extension. If you configure a global command **`access-list dynamic-extended`**, then a user configured with the **`access-enable`** auto-command may re-login to the router to extend the *absolute* timeout by the value configured under the dynamic ACL entry. This allows session prolongation by a user, without terminating any existing connections.

Another option is a manual clear feature. It only works with numbered extended ACLs and allows an administrator to deactivate selected dynamic access-list entries manually. The command syntax is

```
clear access-template <ACL-NUMBER> <DYNAMIC-ENTRY-NAME>
<SRC-IP> <SRC-MASK> <DST-IP> <DST-MASK>
```

For example the following command:

`clear access-template 100 TEST any 10.0.0.0 0.0.0.255`

will clear any dynamic entry cloned from the above template. Note that you do not need to specify the “permit” or “deny” keyword or specify the protocol name.

For verification here, first telnet to R6 directly and log in as “ENABLE” user. This should activate the host entry in the access-list.

```
Rack1SW1#telnet 150.1.6.6
Trying 150.1.6.6 ... Open

User Access Verification

Username: ENABLE
Password: CISCO
[Connection to 150.1.6.6 closed by foreign host]

Rack1R6#show ip access-lists
Extended IP access list 100
    10 permit tcp any any eq telnet (87 matches)
    20 permit tcp any any eq 7001
    30 permit udp any any eq rip (15 matches)
    40 Dynamic ACCESS permit tcp any any eq www
       permit tcp host 155.1.67.7 any eq www
    50 deny ip any any log
```

Now try connecting to the port allowed by the new dynamic entry. Note that as soon as you Telnet through, the inactivity timer starts (300 seconds).

```
Rack1SW1#telnet 150.1.4.4 80
Trying 150.1.4.4, 80 ... Open
GET / HTTP/1.1

WWW-Authenticate: Basic realm="level_15_access"

401 Unauthorized

[Connection to 150.1.4.4 closed by foreign host]
Rack1SW1#

Rack1R6#show ip access-lists
Extended IP access list 100
    10 permit tcp any any eq telnet (87 matches)
    20 permit tcp any any eq 7001
    30 permit udp any any eq rip (24 matches)
    40 Dynamic ACCESS permit tcp any any eq www
       permit tcp host 155.1.67.7 any eq www (12 matches) (time left
295)
    50 deny ip any any log
```

Now clear the access-template:

```
Rack1R6#clear access-template 100 ACCESS host 155.1.67.7 any
Rack1R6#show ip access-lists 100
Extended IP access list 100
    10 permit tcp any any eq telnet (87 matches)
    20 permit tcp any any eq 7001
    30 permit udp any any eq rip (57 matches)
    40 Dynamic ACCESS permit tcp any any eq www
    50 deny ip any any log
```

Try connecting to the rotary line (port 7001). As we remember, it enables the actual dynamic entry and does not insert the host IP address.

```
Rack1SW1#telnet 150.1.6.6 7001
Trying 150.1.6.6, 7001 ... Open


User Access Verification

Password: CISCO
[Connection to 150.1.6.6 closed by foreign host]
Rack1SW1#

Rack1R6#show ip access-list 100
Extended IP access list 100
    10 permit tcp any any eq telnet
    20 permit tcp any any eq 7001 (42 matches)
    30 permit udp any any eq rip (6 matches)
    40 Dynamic ACCESS permit tcp any any eq www
       permit tcp any any eq www
    50 deny ip any any log (1 match)
```

As you can see now anyone can connect across the router to the HTTP port.

☠ Pitfall

Note that you can have only one dynamic entry per access-list. In addition, if you are using dynamic ACLs with AAA enabled, make sure you configure local AAA exec authorization when you bind auto-commands to user names (or use the “none” or “if-authenticated” methods if you bind the command to VTY lines).

11.9 Filtering Traffic with Time-Based Access Lists

- Restrict the users behind R2 from accessing BB1 on weekends.
- Additionally, disallow access to BB1 from 6pm to 9am on the remaining days.

Configuration

```
R2:
!
! Evenings and nights on weekdays.
!
time-range WEEKDAYS_EVES
 periodic weekdays 18:00 to 23:59
 periodic weekdays 0:00 to 8:59
!
! Weekends
!
time-range WEEKENDS
 periodic weekend 0:00 to 23:59
!
ip access-list extended OUTBOUND
 deny ip any any time-range WEEKDAYS_EVES
 deny ip any any time-range WEEKENDS
 permit ip any any
!
interface FastEthernet 0/0
 ip access-group OUTBOUND out
```

Verification

Note

Time-based ACLs allow the use of time ranges, bound to a specific entry. This allows the entry to be active only during the time-range specified. There are two types of time-ranges: absolute and periodic. Absolute time-ranges define non-repeating time intervals (e.g. from Jan 10 to Jan 12). Periodic time-ranges define recurring time-intervals that may repeat infinitely (e.g. every Wednesday from 6am to 11am). The latter type is based on the concept of weekdays: e.g. you may define periods based on days of week (Mon, Tue, Fri etc) and specific ranges like daily, weekends, weekdays (Mon-Fri). Absolute intervals define starting dates/times and ending dates/times.

It is important to note that time-ranges start with the first second of the first minute in the range and end with the last second of the last minute in the range. For example, the interval from 00:01 to 02:00 will last from 00:01:00 to 02:00:59 (hh:mm:ss). Therefore, if you want some interval to end at exactly 4:00pm use the value “15:59” (note that IOS uses the 24 hour time format). Note that the exact boundaries of time ranges may vary based on implementation.

In our scenario, we define two time ranges. The first one covers weekends and the second one covers the time interval from 6pm to 9am every weekday.

As usual, remember that local traffic is not subject to an outbound ACL check. Therefore, try connecting from R3. At first, the connection fails since one of the time-ranges is active:

```
Rack1R3#telnet 192.10.1.254
Trying 192.10.1.254 ...
% Destination unreachable; gateway or host down

Rack1R3#

Rack1R2#show clock
05:14:07.194 UTC Sun Sep 28 2008

Rack1R2#sh ip access-lists
Extended IP access list OUTBOUND
    10 deny ip any any time-range WEEKDAYS_EVES (inactive)
    20 deny ip any any time-range WEEKENDS (active) (2 matches)
    30 permit ip any any
Rack1R2#
```

Now change the time on R2 to Monday during “business hours” and try connecting again:

```
Rack1R3#telnet 192.10.1.254
Trying 192.10.1.254 ... Open
<output omitted>
```

The connection is now successful. Check the access-list counters again:

```
Rack1R2#show ip access-lists
Extended IP access list OUTBOUND
    10 deny ip any any time-range WEEKDAYS_EVES (inactive)
    20 deny ip any any time-range WEEKENDS (inactive) (2 matches)
    30 permit ip any any (12 matches)
```

11.10 Traffic Filtering with Policy Based Routing

- Configure R6 to drop ICMP packets of 100 bytes in size entering the VLAN 146 interface.
- Ensure R6 only drops packets leaving via the VLAN 67 interface.
- The router should not send ICMP unreachable notifications about dropped packets.

Configuration

```
R6:
ip access-list extended ICMP
 permit icmp any any
!
route-map DROP
 match ip address ICMP
 match interface FastEthernet 0/0.67
 match length 100 100
 set interface Null0
!
interface FastEthernet 0/0.146
 ip policy route-map DROP
!
interface Null0
 no ip unreachables
```

Verification

Note

Policy Based Routing allows the implementation of select packet filtering based on various criteria. Among the most important are:

1) Based on any access-list (standard/extended)
2) Based on packet size
3) Based on the packet ToS byte
4) Based on the output interface (allows sub-interface granularity)

The packet drop behavior with PBR is achieved by using **`set interface Null0`** command. Based on the configuration of the Null0 interface, the router may generate an ICMP unreachable message. You can disable this feature to conserve router resources by using the command **`no ip unreachables`** under the Null0 interface. However, the router only sends unreachables for process-switched packets.

For verification, send ICMP packets of different sizes from R4 to SW1. You may need to remove the access-list applied to the R6 VLAN 67 interface:

```
Rack1R4#ping 150.1.7.7 size 200

Type escape sequence to abort.
Sending 5, 200-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Rack1R4#ping 150.1.7.7 size 100

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
Rack1R4#

Rack1R6#show route-map
route-map DROP, permit, sequence 10
  Match clauses:
    ip address (access-lists): ICMP
    interface FastEthernet0/0.67
    length 100 100
  Set clauses:
    interface Null0
  Policy routing matches: 5 packets, 590 bytes
```

Note that the byte count is 590 not 500. The counters actually account for Layer 2 overhead, which is 18 bytes for 802.1q tagged frames.

Now verify that R6 sends IP unreachables if you enable them back on the Null0 interface and disable CEF switching on the VLAN 146 interface.

```
R6:
interface Null 0
 ip unreachables
!
interface FastEthernet 0/0
 no ip route-cache

Rack1R4#ping 150.1.7.7 size 100

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
U.U.U
Success rate is 0 percent (0/5)
```

11.11 Preventing Packet Spoofing with uRPF

- R4 connects to your ISP at the border of your network.
- Ensure that R4 does not accept packets with IP addresses of the internal subnets on its connection to the ISP.
- There is another connection to the same ISP in the network, so account for possible asymmetric routing issues on R4.
- At the same time, R4 should only accept packets from legitimate subnets of 150.X.0.0/16 and 155.X.0.0/16 learned via IGP on its internal connections.
- Log all packets with spoofed sources.

Configuration

```
R4:
!
! Access-List to deny and log any violating packet
!
access-list 100 deny ip any any log
!
! Apply Loose uRPF to the upstream link
!
interface FastEthernet 0/0
ip verify unicast source reachable-via any 100
!
! Apply strict uRPF to all internal interfaces
!
interface FastEthernet 0/1
 ip verify unicast source reachable-via rx 100
!
interface Serial 0/0
 ip verify unicast source reachable-via rx  100
!
interface Serial 0/1
 ip verify unicast source reachable-via rx  100
```

Verification

Note

uRPF or Unicast Reverse Path Forwarding is the concept of verifying the routing path for the source IP address found in an IP packet. Usually routers only use destination IP addresses to look up the next hop for an IP packet. Reverse-Path Forwarding, however, is used with multicast routing. From a security perspective, uRPF is useful for checking IP address spoofing conditions.

Generally, packets arrive on the interfaces that are on the shortest path to the source of the packets. This is the natural result of the IGP routing protocol at work on the device. However, with IP spoofing attacks, a malicious user may inject packets with IP addresses not belonging to its segment or network area. From the router perspective, such packets may appear on the interfaces not on the shortest path to their source. You should note that packets may have spoofed sources and appear on the shortest-path interface in cases when you have just one connection to all destinations.

Based on its function, uRPF could be a very useful security feature to prevent spoofing attacks, especially on routers that have a diverse number of connections, for example a system at an Internet peering point. The feature has two general modes of operation:

1) Strict mode - when you enable uRPF on an interface with the **`ip verify unicast source reachable-via rx`** command (or the legacy format **`ip verify unicast reverse-path`**), the router applies the uRPF check to the source IP addresses of incoming packets. The source IP address must match an explicit IP route in the routing table, and the next hop for this entry should point out of the interface the packet was received from. Otherwise, the router will drop the packets.

2) Loose mode - you enable the loose mode with the command **`ip verify unicast source reachable-via any`**. When a router receives a packet on the interface with loose uRPF enabled, it just checks that it has an IP route matching the source address in the packet. It does not matter whether the next hop for this route points out the receiving interface or not. The exception to this checking is that if the route is to Null0, the packet is dropped. This feature is useful in enterprises that have more then one ISP uplink and use asymmetric routing. With asymmetric routing, packets may take paths not allowed with strict uRPF.

uRPF does have additional features. The first one is uRPF exemptions and violation logging. With this feature, you may specify a standard or extended access-list as follows: **`ip verify unicast source reachable-via {rx|any} <ACL-NUM>`**. The uRPF feature consults this access-list for packets *violating* the uRPF condition. If the ACL permits a packet, it is allowed to pass through. If the ACL denies the packet, the router drops it. You may use the **`log`** keyword to log the packets allowed or denied by the uRPF access-list.

The other two features are:

1) Allowing the router to do a self-ping with strict uRPF enabled on the interfaces (i.e. accepting packets sourced from the router on the interface they were sent from).

2) Allowing the use of a default route with strict uRPF. In this case, the router will match source IP addresses against the default-route entry (0.0.0.0/0) as well. This is useful when you want to accept all packets on your Internet connections, but feature protection from spoofed packets coming from your internal network.

To verify the feature, first make sure it has been applied to all interfaces:

```
Rack1R4#show ip interface fastEthernet 0/1 | inc verify
  IP verify source reachable-via RX, ACL 100

Rack1R4#show ip interface fastEthernet 0/0 | inc verify
  IP verify source reachable-via ANY, ACL 100

Rack1R4#show ip interface Serial 0/0 | inc verify
  IP verify source reachable-via RX, ACL 100

Rack1R4#show ip interface Serial 0/1 | inc verify
  IP verify source reachable-via RX, ACL 100
```

Now generate some spoofed traffic from R1 (the inside network) across R4. Create a special Loopback interface on R1 to accomplish this:

```
R1:
interface Loopback1
 ip address 150.2.1.1 255.255.255.0

R4:
ip access-list log-update threshold 1

Rack1R1#ping 150.1.4.4 source loopback 1 repeat 10

Type escape sequence to abort.
Sending 10, 100-byte ICMP Echos to 150.1.4.4, timeout is 2 seconds:
Packet sent with a source address of 150.2.1.1
```

Observe the logging and statistics on R4:

```
Rack1R4#
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 150.2.1.1 -> 150.1.4.4
(0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 150.2.1.1 -> 150.1.4.4
(0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 150.2.1.1 -> 150.1.4.4
(0/0), 1 packet

Rack1R4#show ip interface fastEthernet 0/1
<snip>
  IP verify source reachable-via RX, ACL 100
  10 verification drops
  0 suppressed verification drops

Rack1R4#show ip access-lists 100
Extended IP access list 100
    10 deny ip any any log (10 matches)
```

11.12 Using NBAR for Content-Based Filtering

- Configure R6 to drop any potentially dangerous HTTP downloads.
- The dangerous files have extensions ".exe", ".com", or ".bin".
- Use a single pattern to match all the filenames at once.

Configuration

```
R6:
class-map match-all EXTENSION
 match protocol http url "*.bin|*.exe|*.com"
!
!
policy-map DROP
 class EXTENSION
  drop
!
interface FastEthernet 0/0.146
  service-policy output DROP
```

Verification

Note

The NBAR (Network Based Application Recognition) protocol classification mechanism allows deep inspection for some protocol types. One notable feature is matching various fields inside HTTP headers. In recent IOS versions, this feature evolved into the full-blown Flexible Packet Matching (FPM) feature, which allows for comprehensive inspection of any IP packets.

Commonly you see HTTP inspection used for matching the URL field against certain patterns. The URL matching occurs for HTTP GET/PUT/POST requests. It is important to note that the NBAR engine classifies bi-directional traffic flows. That is, if you apply a policy map matching the URL pattern inbound on an interface, the policy map will classify both incoming HTTP requests as well as packets returning in server replies. This is because NBAR classification occurs independently of policy-map direction, and the classification decision applies to both parts of a bi-directional traffic flow.

The command to match a URL is:

`match http protocol url <pattern>`

The URL matching feature uses special wildcard symbols:

"*" – matches any sequence of characters (non-empty)
"?"– matches any single character (you need to press Ctrl-V in order to enter "?")
"|" – alternative, logical OR
"[]" – range, e.g. [ab] matches either "a" or "b"
"()" – grouping; delimit the logical end of a pattern; or example, you can use "*.(exe|bin)" as equivalent to "*.exe|*.bin"

All matching is case insensitive. The pattern "text" matches "TEXT" as well. The engine matches your URL pattern against the directory path and the file name in the URL. For example, if the URL string is "http://www.cisco.com/pub/uploads/image.jpeg", the matching procedure will only use the "pub/uploads/image.jpeg" part of the URL. When you submit a request like the above URL, it translates into the following headers (there are actually more, but this is the bare minimum):

```
GET /pub/uploads/image.jpeg HTTP/1.1
Host: www.cisco.com
```

Thus if you need to match a host name, you should use the **`match protocol http host`** statement. You can use the same wildcard characters to match the HTTP Host header.

We are going to use the following single-line expression for our task:

"*.bin|*.exe|*.com"

This pattern matches files with extension "*.com", "*.exe" or "*.bin". You can also use the "match-any" class map and match multiple URL strings on separate lines, resulting in the same effect as separating a pattern with the symbol "|".

The second interesting part of using this feature is the MQC **`drop`** operation. Under any MQC class, you can apply this operation to drop the matching packets.

For verification of our task, simulate an HTTP server with SW1 and create some files in the flash with extensions “exe”, “com”, and “bin”.

```
SW1:
ip http server
ip http path flash:
!
Rack1SW1#copy flash:vlan.dat flash:vlan.exe
Destination filename [vlan.exe]?
Copy in progress...C
1216 bytes copied in 1.082 secs (1124 bytes/sec)

Rack1SW1#copy flash:vlan.dat flash:vlan.com
Destination filename [vlan.com]?
Copy in progress...C
1216 bytes copied in 0.025 secs (48640 bytes/sec)

Rack1SW1#copy flash:vlan.dat flash:vlan.BIN
Destination filename [vlan.BIN]?
Copy in progress...C
1216 bytes copied in 0.025 secs (48640 bytes/sec)
```

Now try downloading those files on R1 across R6:

```
Rack1R1#copy http://admin:cisco@150.1.7.7/vlan.exe null:
%Error opening http://admin:cisco@150.1.7.7/vlan.exe (I/O error)

Rack1R6#show policy-map interface fastEthernet 0/0.146
 FastEthernet0/0.146

  Service-policy output: DROP

    Class-map: EXTENSION (match-all)
      9 packets, 1827 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http url "*.bin|*.exe|*.com”
      drop

    Class-map: class-default (match-any)
      15 packets, 2210 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

Rack1R1#copy http://admin:cisco@150.1.7.7/vlan.BIN null:
%Error opening http://admin:cisco@150.1.7.7/vlan.BIN (I/O error)

Rack1R1#copy http://admin:cisco@150.1.7.7/vlan.com null:
%Error opening http://admin:cisco@150.1.7.7/vlan.com (I/O error)
```

```
Rack1R6#show policy-map interface fastEthernet 0/0.146
 FastEthernet0/0.146

  Service-policy output: DROP

    Class-map: EXTENSION (match-all)
      52 packets, 9884 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http url “*.bin|*.exe|*.com”
      drop

    Class-map: class-default (match-any)
      57 packets, 11676 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Now attempt to transfer a legitimate file and ensure that R6 allows it to pass through:

```
Rack1R1#copy http://admin:cisco@150.1.7.7/vlan.dat null:
Loading http://***********@150.1.7.7/vlan.dat !
1216 bytes copied in 0.037 secs (32865 bytes/sec)
```

11.13 TCP Intercept

- Configure R6 to protect the WWW servers on VLAN 146 from a SYN flood attack.
- Intercept connections as they go to the servers and close connections after 30 seconds of inactivity.
- Allow for a maximum of 100 semi-established connections and drop their number down to 80 when the threshold is crossed.
- The router should start dropping half-open connections when they cross the rate threshold of one connection per second. The router should not stop until the average rate is one per two minutes.
- The router should not account for connection age when dropping them.

Configuration

```
R6:
access-list 146 permit ip any 155.1.146.0.0.0.255
ip tcp intercept list 146
ip tcp intercept connection-timeout 30
ip tcp intercept max-incomplete low 80
ip tcp intercept max-incomplete high 100
ip tcp intercept one-minute low 30
ip tcp intercept one-minute high 60
ip tcp intercept drop-mode random
```

Verification

Note

The TCP Intercept feature protects against TCP SYN flood attacks targeted at servers located behind the router. The idea of SYN-flooding is to send a huge number of SYN packets towards the server, without ever finishing the 3-way TCP handshake. The server then exhausts its resources and may eventually refuse to accept a connection from a legitimate user.

There are two modes of the TCP Intercept feature. In the first mode, the router acts as a proxy between clients and the server. For every incoming SYN packet, the router immediately answers with a SYN-ACK packet to the client and sends a SYN packet to the server. The router then waits for a final ACK from the client and a SYN-ACK from the server. When both “parts” of the connection establish, the router joins them and routes packets transparently.

If one of the endpoints does not respond, the router probes it repeatedly using an exponential backoff algorithm. The router sends the second SYN packet in 1 second; 3rd SYN in 2 seconds, 4th SYN in 4 seconds, 5th retransmit in 8 seconds and 6th retransmit in 16 seconds. Overall, this takes 1+2+4+8+16=31 seconds.

You can enable the TCP intercept mode using the command:

```
ip tcp intercept mode intercept
```

In this mode the router limits the total number of half-open (3-way handshake unfinished) connections from clients. The limit is set using the command:

```
ip tcp intercept max-incomplete {low|high} <Thresh>
```

This limit only applies to the half-open connections and does not affect the established connections. If there is a new connection forming and the number reaches the high threshold, the router starts dropping the half-open connections until their number falls down to the low threshold.

You may also limit the rate of the incoming SYN-packets using a per-minute value. Specifically, you can set the maximum number of half-open sessions allowed per-minute using the command:

```
ip tcp intercept one-minute {low|high} <Thresh>
```

When the number of semi-established sessions opened during the last minute exceeds the high threshold, the router starts dropping them until the measured rate drops below the low threshold. The router keeps track of both thresholds at the same time, and drops the connections until their number reaches the low threshold.

Depending on the drop-mode, the router may drop either the oldest half-open session or any random session (until it reaches the low threshold).

```
ip tcp intercept drop-mode {random|oldest}
```

In addition to deleting the excessive connection states, the router also changes the backoff algorithm behavior. It reduces all retransmit timers in half, so the additional probes are sent in 0.5 seconds, 1 second, 2 seconds, 4 seconds, and 8 seconds. This totals to 15 seconds of probing for a non-answering endpoint. Altogether, dropping the connections and reducing the backoff timers is called “Aggressive” mode behavior.

The other settings related to the TCP Intercept mode are the following:

1) FIN/RST timeout - this is the amount of time the router keeps a connection entry after it receives a FIN packet from a client or server. After this amount of time, the router sends forceful RST packets to the server (making it free the resources) and removes the entry. The command to set this timeout is:

```
ip tcp intercept finrst-timeout <Timeout>
```

2) Limit the scope of TCP interception:

```
ip tcp intercept list <ACL-NUM>|<ACL-NAME>
```

Only the connections matching the access-list are subject to TCP interception. By default, TCP Intercept does not apply to any transit connection, so make sure you specified a list of packets to match.

3) The timeout for inactive connections - defines the amount of time that TCP Intercept maintains an inactive connection (no data in it):

```
ip tcp intercept connection-timeout <TIMEOUT>
```

After the timeout expires, the router sends RST packets to both sides of the connection, effectively terminating it.

For verification in our task, try connecting to R4 (a legitimate connection) across R6 and keep the connection inactive for 30 seconds:

```
Rack1R6#debug ip tcp intercept
TCP intercept debugging is on

Rack1SW1#telnet 155.1.146.4
Trying 155.1.146.4 ... Open


User Access Verification

Password: cisco
```

```
Rack1R6#
05:27:09: INTERCEPT: new connection (155.1.67.7:11008 SYN ->
155.1.146.4:23)
05:27:09: INTERCEPT(*): (155.1.67.7:11008 <- ACK+SYN 155.1.146.4:23)
05:27:09: INTERCEPT: 1st half of connection is established
(155.1.67.7:11008 ACK -> 155.1.146.4:23)
05:27:09: INTERCEPT(*): (155.1.67.7:11008 SYN -> 155.1.146.4:23)
05:27:09: INTERCEPT: 2nd half of connection established
(155.1.67.7:11008 <- ACK+SYN 155.1.146.4:23)
```

First, the router adds the connection to the intercept list and ensures that both parts of the connections (client <-> router and router <-> server) have been established. After that, the router counts the time of inactivity.

```
05:27:09: INTERCEPT(*): (155.1.67.7:11008 ACK -> 155.1.146.4:23)
05:27:09: INTERCEPT(*): (155.1.67.7:11008 <- WINDOW 155.1.146.4:23)
05:27:41: INTERCEPT: ESTAB timing out (155.1.67.7:11008 <->
155.1.146.4:23)
```

After 30 seconds, the router terminates the connection by sending RST packets to both the client and the server.

```
05:27:41: INTERCEPT(*): (155.1.67.7:11008 <- RST 155.1.146.4:23)
05:27:41: INTERCEPT(*): (155.1.67.7:11008 RST -> 155.1.146.4:23)
```

Now we block ACK packets going from the client to the server, effectively preventing the established connection formation.

```
R6:
ip access-list extended NO_ACK
 deny tcp any any established
 permit ip any any
!
interface FastEthernet 0/0.67
 ip access-group NO_ACK in
```

Open a connection to R4 from SW1. It seems to be open, because R4 responds with a SYN+ACK.

```
Rack1SW1#telnet 155.1.146.4
Trying 155.1.146.4 ... Open

05:36:02: INTERCEPT: new connection (155.1.67.7:11009 SYN ->
155.1.146.4:23)
```

However, the last ACK from SW1 is blocked by R6. Therefore, R6 tries to re-send the SYN+ACK to SW1 five times, and gives up after 31 seconds. After that, it sends the RST and terminates the connection.

```
05:36:02: INTERCEPT(*): (155.1.67.7:11009 <- ACK+SYN 155.1.146.4:23)
05:36:03: INTERCEPT(*): SYNRCVD retransmit 1 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:05: INTERCEPT(*): SYNRCVD retransmit 2 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:09: INTERCEPT(*): SYNRCVD retransmit 3 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:17: INTERCEPT(*): SYNRCVD retransmit 4 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:33: INTERCEPT: SYNRCVD retransmitting too long (155.1.67.7:11009
<-> 155.1.146.4:23)
05:36:33: INTERCEPT(*): (155.1.67.7:11009 <- RST 155.1.146.4:23)
```

11.14 TCP Intercept Watch Mode

- Modify your TCP intercept configuration in order to reset any connection that does not reach the established state in 20 seconds.

Configuration

```
R6:
ip tcp intercept mode watch
ip tcp intercept watch-timeout 20
```

Verification

Note

TCP Intercept Watch Mode is a “lightweight” version of the full TCP Intercept Mode. Instead of sitting in the path of TCP connections, the router just watches them as they go. The major timeout value for the watch-mode is the “watch timeout”. If the client and server did not negotiate a connection during this time, the router forcefully injects RST messages to the server, causing it to free up the allocated resources. Note that this is in contrast to the intercept mode, where the router attempts communicating with the client and the server by itself, using an exponential backoff. The command is:

`ip tcp intercept watch-timeout <TIMEOUT>.`

This command only makes sense when you configure the Watch mode using the command:

`ip tcp intercept mode watch`

Watch mode utilizes the same timers as the Intercept mode. The semi-established connection thresholds are treated in the same way, switching to Aggressive mode when their number reaches the high threshold. When in Aggressive mode, the configured watch timeout is reduced by half as well. There are no retransmission attempts, as the router is not acting as a proxy.

To verify this task, configure R6 to block the ACK packets from SW1:

```
R6:
ip access-list extended NO_ACK
 deny tcp any any established
 permit ip any any
!
interface FastEthernet 0/0.67
 ip access-group NO_ACK in
```

Now enable TCP Intercept debugs and try connecting from SW1 to R4:

```
Rack1R6#debug ip tcp intercept
TCP intercept debugging is on

Rack1SW1#telnet 155.1.146.4
Trying 155.1.146.4 ... Open
```

R6 does not intervene with the connection, but passively watches it. As the watch timeout passes (20 seconds), R6 sends a RST to the server and terminates the connection, freeing resources on the server:

```
Rack1R6#
06:23:48: INTERCEPT: new connection (155.1.67.7:11010 SYN ->
155.1.146.4:23)
06:23:48: INTERCEPT: (155.1.67.7:11010 <- ACK+SYN 155.1.146.4:23)
06:23:50: INTERCEPT: server packet passed in SYNRCVD (155.1.67.7:11010
<- 155.1.146.4:23)
06:23:54: INTERCEPT: server packet passed in SYNRCVD (155.1.67.7:11010
<- 155.1.146.4:23)
06:24:02: INTERCEPT: server packet passed in SYNRCVD (155.1.67.7:11010
<- 155.1.146.4:23)
06:24:08: INTERCEPT: SYNRCVD timing out (155.1.67.7:11010 <->
155.1.146.4:23)
06:24:08: INTERCEPT(*): (155.1.67.7:11010 RST -> 155.1.146.4:23)
```

Open a connection to R4 from SW1. It seems to be open, because R4 responds with a SYN+ACK. However, the last ACK from SW1 is blocked by R6. Therefore, R6 tries to re-send the SYN+ACK to SW1 five times, and gives up after 31 seconds. After that, it sends a RST and terminates the connection.

```
05:36:02: INTERCEPT(*): (155.1.67.7:11009 <- ACK+SYN 155.1.146.4:23)
05:36:03: INTERCEPT(*): SYNRCVD retransmit 1 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:05: INTERCEPT(*): SYNRCVD retransmit 2 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:09: INTERCEPT(*): SYNRCVD retransmit 3 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:17: INTERCEPT(*): SYNRCVD retransmit 4 (155.1.67.7:11009 <-
ACK+SYN 155.1.146.4:23)
05:36:33: INTERCEPT: SYNRCVD retransmitting too long (155.1.67.7:11009
<-> 155.1.146.4:23)
05:36:33: INTERCEPT(*): (155.1.67.7:11009 <- RST 155.1.146.4:23)
```

11.15 Packet Logging with Access-Lists

- Configure R6 to log all ICMP packets entering its VLAN 146 interface.
- Aggregate logging messages so that a log entry is generated after five access-list entry hits.
- Reduce the burden on the router CPU by process switching only one packet per second.
- Ensure the logged messages allow you to track the MAC address of the host sending the packet.

Configuration

```
R6:
ip access-list log-update threshold 5
ip access-list logging interval 1000
!
ip access-list extended LOGGING
 permit icmp any any log-input
 permit ip any any
!
interface FastEthernet0/0.146
 ip access-group LOGGING in
```

Verification

Note

You can configure an access-list entry with the **`log`** keyword, making it log the matching packets via syslog. Using the **`log-input`** keyword, you can also log the input interface and the source MAC address. There are two configuration “knobs” to be used with access-list based logging. The first one is the logging update-threshold. It specifies how many times a packet must hit the access-list entry to generate a logging message. By default, this value is zero, which means the router would generate an entry based on a periodic timeout of 5 minutes. This command serves the purpose of aggregating the hits on the entry and the format is:

`ip access-list log-update threshold <HIT-COUNT>`

When this value is non-zero, the router generates a log entry for every number of hits and also produces the periodic 5 minute logging message.

The second knob relates to the behavior of packet logging. You should remember that every logged packet is process-switched, thus a large packet flow may easily eat all your CPU resources. For this reason, you may want to rate-limit the amount of process-switched packets using the command:

`ip access-list logging interval <TIME-INTERVAL>`

This command allows only one packet per the interval to be process-switched. All exceeding packets are not process-switched and thus are not accounted for logging purposes. By default, this feature is off and all packets are process-switched. Note that this interval does not apply to packets destined to the router itself, since these are process-switched by default. Using this command limits the effect of packet logging on the CPU, but may result in an unpredictable number of packets being logged. In reality it is useful when you just need a general hint of packet matches, not the detailed statistics.

You may also limit the amount of all syslog messages (including the ACL logging messages) using the generic syslog rate-limiting feature.

To verify, try sending a barrage of ping packets from R4 to SW1 across R6:

```
Rack1R4#ping 150.1.7.7 repeat 1000

Type escape sequence to abort.
Sending 1000, 100-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<snip>
Success rate is 100 percent (1000/1000), round-trip min/avg/max =
1/2/12 ms
Rack1R4#

Rack1R6#
07:19:15: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
```

Now change the logging interval and set the log-update threshold to 1 packet:

```
R6:
ip access-list logging interval 1
ip access-list log-update threshold 1

Rack1R4#ping 150.1.7.7 repeat 1000 size 1500

Type escape sequence to abort.
Sending 1000, 1500-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!<
<snip>
Success rate is 100 percent (1000/1000), round-trip min/avg/max =
4/5/12 ms
```

```
Rack1R6#
07:31:35: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
07:31:38: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
07:31:38: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
07:31:39: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
07:31:39: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
07:31:40: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 1 packet
```

As you can see, in both cases the source MAC address and the input interface were logged. To get the true count of aggregate packet matches, disable the logging interval:

```
Rack1R4#ping 150.1.7.7 repeat 50 size 1500

Type escape sequence to abort.
Sending 50, 1500-byte ICMP Echos to 150.1.7.7, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (50/50), round-trip min/avg/max = 4/7/12 ms
Rack1R4#

Rack1R6#show logging
<snip>

07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
07:37:27: %SEC-6-IPACCESSLOGDP: list LOGGING permitted icmp 155.1.146.4
(FastEthernet0/0.146 0011.21c8.ec41) -> 150.1.7.7 (0/0), 5 packets
```

Now you see 10 logging entries with 5 packet counts each.

11.16 Stateful Filtering with CBAC

- Configure R4 as a stateful firewall. R4's VLAN 146 interface is the protected network and the Serial interface between R4 and R5 is the connection to outside networks.
- Ensure that TFTP and FTP protocols function across the firewall.
- Collect connection statistics for HTTP transfers.
- Allow pinging across the firewall but do not add any special ACL entries to accomplish this.
- Allow UDP sessions across the firewall with a 30 second inactivity timeout.
- Ensure that inspection entries for DNS requests expire in 10 seconds but do not inspect the DNS protocol for this.
- Disable CBAC alerts for all protocols with the exception of FTP.
- Do not allow any other TCP-based protocols except for those mentioned above.
- In the future some users are going to use the Cisco ezVPN client from inside the firewall. Ensure you do not need to add any ACL entries in order to allow ezVPN connections across the firewall.

Configuration

```
R4:
!
! Disable alerts globally and configure DNS timeout
!
ip inspect alert-off
ip inspect dns-timeout 10
!
! Define inspection rule and protocol-specific settings
! Enable alerts for FTP only
!
ip inspect name INSPECT ftp alert on
!
! Enable audit trails for HTTP only
!
ip inspect name INSPECT http audit-trail on
!
! Inspect all UDP sessions and expire them after 30 seconds
! of inactivity
!
ip inspect name INSPECT udp timeout 30
ip inspect name INSPECT tftp
!
! Inspecting ISAKMP dynamically opens a hole for IPsec Phase 2 traffic
!
ip inspect name INSPECT isakmp
ip inspect name INSPECT icmp
!
```

```
ip access-list extended INBOUND
 permit udp any any eq 520
 deny ip any any
!
interface Serial 0/1
 ip inspect INSPECT out
 ip access-group INBOUND in
```

Verification

Note

CBAC or context-based access-control introduces true stateful inspection to IOS code. You can compare it to the reflexive ACL feature as being an advanced inspection technology. The general idea is to inspect protocol-specific information in traffic flows going across the router and dynamically open holes in access-lists for returning traffic. CBAC configurations commonly consist of two parts:

1) The CBAC inspection rule (defines traffic classes to inspect) applied to an interface in the direction matching the initial flow of traffic (usually from the protected network to the outside world).

2) An access-group applied to an interface in the direction matching the returning traffic flow (usually from the outside network to the protected network).

Commonly, the initial flow of the traffic is from the protected network (inside of the firewall) to the unprotected network (outside of the firewall). For example, think of an office network initiating HTTP connections to the Internet across the filtering router.

You define an inspection rule using the command:

```
ip inspect name <RULE-NAME> <protocol> …
ip inspect name <RULE-NAME> <protocol> …
```

and apply it to an interface using the interface-level command

```
ip inspect <RULE-NAME> {in|out}
```

The rule inspects traffic flowing in the direction specified and installs a special ACL bypass entry in the packet switching path to allow returning packets to pass through the router, even if there is an ACL denying them (in older IOS releases CBAC was simply adding new ACL entries dynamically, but now it's a bypass rule which performs more efficiently). Therefore, even if there is a local ACL in the path of returning packets, it will be bypassed by packets identified as part of the returning protocol traffic flow. For example, with the FTP protocol, CBAC dynamically permits the active data transfer connection (which is in the reverse direction), even without knowing the communication port in advance.

You may put the inspection rule and the access-list on different interfaces. As long as they apply to the initial connections and returning traffic flows respectively it's fine. For example, you can apply the inspection rule as outgoing on the outside interface and an access-list in the incoming direction. You can also put the inspection rule in the incoming direction on the inside interface and the access-list in incoming direction on the outside interface.

CBAC protocol inspection can provide you with alarms, notifying you of protocol violations and potential attacks. By default alerts are enabled for all inspected protocols. You can disable alarms globally and configure them selectively per-protocol.

In addition to providing you with alarm messages, CBAC can perform basic traffic accounting (duration, bytes transferred) by logging audit trails in syslog. This feature is disabled by default, and could be enabled globally for all inspected protocols or tuned selectively per protocol in an inspection rule.

CBAC may inspect generic TCP/UDP connections (without looking for specific upper level protocol information) just to check their integrity and open a hole for returning traffic. If you enable TCP inspection along with HTTP inspection at the same time, HTTP inspection would be used for connections made on port 80.

By default, CBAC (as well as outgoing ACLs) do not apply to router-generated traffic. You need to add manual ACL entries in order to permit returning traffic for router-generated packets. Alternatively, you can use local policy routing or a special CBAC feature for router-generated traffic inspection. We are going to discuss this feature in a separate task.

CBAC has a number of configurable inspection timeouts. In this task we are going to discuss just the application inactivity timeout. For every inspected protocol you can set up an inactivity timeout. CBAC will close the inspection entry after the timeout expires.

There is a special command **`ip inspect dns-timeout <TIMEOUT>`** which applies when you inspect generic UDP sessions. Since DNS represents a special case of the UDP protocol, it has a separate timeout. DNS “sessions” are quick request/response, so by default when CBAC sees UDP packet on port 53 it holds the inspection entry for 5 seconds, not the default interval specified for the UDP protocol. In the later releases of IOS, you can define separate DNS inspection rules with their own timeouts.

Check the general information on the configured inspection rules. You can use show commands to see the timeout values (also you can use them to learn other default values).

```
Rack1R4#show ip inspect config
Session audit trail is disabled
Session alert is disabled
one-minute (sampling period) thresholds are [400:500] connections
max-incomplete sessions thresholds are [400:500]
max-incomplete tcp connections per host is 50. Block-time 0 minute.
tcp synwait-time is 30 sec -- tcp finwait-time is 5 sec
tcp idle-time is 3600 sec -- udp idle-time is 30 sec
dns-timeout is 10 sec
Inspection Rule Configuration
 Inspection name INSPECT
    ftp alert is on audit-trail is off timeout 3600
    http alert is off audit-trail is on timeout 3600
    dns alert is off audit-trail is off timeout 30
    tftp alert is off audit-trail is off timeout 30
    isakmp alert is off audit-trail is off timeout 30
    icmp alert is off audit-trail is off timeout 10
    udp alert is off audit-trail is off timeout 30

Rack1R4#show ip inspect interfaces
Interface Configuration
 Interface Serial0/1
  Inbound inspection rule is not set
  Outgoing inspection rule is INSPECT
    ftp alert is on audit-trail is off timeout 3600
    http alert is off audit-trail is on timeout 3600
    dns alert is off audit-trail is off timeout 30
    tftp alert is off audit-trail is off timeout 30
    isakmp alert is off audit-trail is off timeout 30
    icmp alert is off audit-trail is off timeout 10
    udp alert is off audit-trail is off timeout 30
  Inbound access list is INBOUND
  Outgoing access list is not set
```

To verify the functionality of CBAC, you can use the following commands

```
Rack1R1#telnet 155.1.45.5 80
Trying 155.1.45.5, 80 ... Open

Rack1R4#show ip inspect sessions
Established Sessions
 Session 852BD3D4 (155.1.146.1:42652)=>(155.1.45.5:80) http SIS_OPEN
Rack1R4#

[Resuming connection 1 to R1 ... ]

Rack1R1#
GET / HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Thu, 02 Oct 2008 12:14:39 GMT
Server: cisco-IOS
Connection: close
Accept-Ranges: none

400 Bad Request

[Connection to 155.1.45.5 closed by foreign host]

Rack1R4#
CBAC: Finding pregen session for src_tableid:0, src_addr:155.1.146.1,
src_port:42652, dst_tableid:0, dst_addr:155.1.45.5, dst_port:80
%FW-6-SESS_AUDIT_TRAIL_START: Start http session: initiator
(155.1.146.1:42652) -- responder (155.1.45.5:80)
%FW-6-SESS_AUDIT_TRAIL: Stop http session: initiator
(155.1.146.1:42652) sent 18 bytes -- responder (155.1.45.5:80) sent 141
bytes
```

The above output demonstrates open HTTP session inside the CBAC tables, as well as audit trails generated for our simulated HTTP connection.

Now you can verify that TFTP is working across the stateful firewall. TFTP is a protocol that uses dynamically generated ports for data transfer, so it is a perfect candidate for CBAC inspection testing.

Configure R5 as a TFTP server and try transferring a file across the firewall:

```
R5:
tftp-server flash:c2600-adventerprisek9-mz.124-10.bin alias test

Rack1R1#copy tftp://155.1.45.5/test null:
Accessing tftp://155.1.45.5/test...
Loading test from 155.1.45.5 (via FastEthernet0/0): !

Rack1R4#show ip inspect sessions
Established Sessions
 Session 852BCC54 (155.1.45.5:49511)=>(155.1.146.1:49822) tftp-data
SIS_OPEN
Half-open Sessions
 Session 852BD154 (155.1.146.1:51100)=>(155.1.45.5:69) tftp SIS_OPENING
 Session 852BD654 (155.1.146.1:49564)=>(155.1.45.5:69) tftp SIS_OPENING
 Session 852BCED4 (155.1.146.1:49822)=>(155.1.45.5:69) tftp SIS_OPENING
 Session 852BD3D4 (155.1.146.1:57759)=>(155.1.45.5:69) tftp SIS_OPENING
```

Note that the initial connection appears as a UDP half-open session. This is because TFTP never replies directly to the original source port from port 69, but rather starts additional data connections. However, overall TFTP works across the firewall, even though it uses dynamic ports.

Finally, test that ping operations work across the firewall by the virtue of ICMP inspection:

```
Rack1R1#ping 155.1.45.5

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.45.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/29/32 ms
```

11.17 Advanced CBAC Features

- Configure R4 so that you do not have to create manual openings for router-generated UDP traffic in the inbound access-lists.
- You expect no more than 5000 concurrent sessions in the busiest hours. Optimize CBAC performance to handle this configured maximum number of sessions.
- Only allow Java applets downloaded from the WWW servers in the subnet 150.X.5.0/24.
- Someone runs an FTP server listening on port 80. The IP address of the server is 150.X.5.25. Ensure that CBAC correctly inspects FTP sessions to this host.
- Ensure that Java filtering applies to connections through an outside HTTP proxy using port 8080.

Configuration

```
R4:
!
! Enable inspection of router-generated UDP traffic (includes RIP)
!
ip inspect name INSPECT udp router-traffic
!
! Increase the hash-table size twice
!
ip inspect hashtable-size 2048
!
! A list of servers allowed for java applet download
!
access-list 99 permit 150.1.5.0 0.0.0.255
!
! Block java applets from all servers with except to the list
!
ip inspect name INSPECT http java-list 99
!
! The server that runs FTP on port 80
!
access-list 98 permit 150.1.5.25
!
! Use port 80 for FTP application on the above server
!
ip port-map ftp port 80 list 98
!
! Inspect HTTP protocol on port 8080 as well
!
ip port-map http port 8080
```

Verification

Note

CBAC uses an internal hash table to store connection information. Every active connection has an entry there. Entries are mapped into hash buckets using a special hash function. The smaller the hash table size (in buckets), the more connection entries map (collide) in the same bucket. On a very busy firewall this may lead to performance degradations, since the lookup through a bucket uses a linear search.

The default hash table size is 1024 buckets. It may vary in size from 2048, 4096 or 8192 buckets. When the number of concurrent connections across the firewall exceeds 4000, you may want to adjust the default size to match approximately half of the maximum connections count. In our case with 5000 sessions, the optimal hash table size would be 2048 buckets.

Java blocking is a special feature that allows the CBAC HTTP inspection engine to block Java applet downloads from certain sites. You specify a list of allowed sites using a standard ACL. This ACL lists servers that users may use to download Java applets.

CBAC uses a global port-mapping table that maps applications to their respective TCP/UDP port numbers. The IOS features a default system mapping for every application that CBAC may inspect. You cannot delete a system defined port mapping, but you can extend it by adding new ports. You may also use an access-list to limit the scope of new port definitions. Port overloading with an access-list applies to sessions destined to servers matching the standard access-list. While you cannot assign a system-defined port to any other application (for example, you cannot assign port 80 to FTP application) you can overload a system-defined port to another application using an access-list. In this scenario, we assign port 80 to an FTP application bound to a particular IP address.

These changes can be verified as follows:

```
Rack1R4#show ip inspect all
Session audit trail is disabled
Session alert is disabled
one-minute (sampling period) thresholds are [400:500] connections
max-incomplete sessions thresholds are [400:500]
max-incomplete tcp connections per host is 50. Block-time 0 minute.
tcp synwait-time is 30 sec -- tcp finwait-time is 5 sec
tcp idle-time is 3600 sec -- udp idle-time is 30 sec
dns-timeout is 10 sec
Inspection Rule Configuration
 Inspection name INSPECT
```

```
    ftp alert is on audit-trail is off timeout 3600
    http java-list 99 alert is off audit-trail is on timeout 3600
    dns alert is off audit-trail is off timeout 30
    tftp alert is off audit-trail is off timeout 30
    isakmp alert is off audit-trail is off timeout 30
    icmp alert is off audit-trail is off timeout 10
    udp alert is off audit-trail is off timeout 30
 inspection of router local traffic is enabled
 Inspection name TEST
    http alert is off audit-trail is off timeout 30

Interface Configuration
 Interface Serial0/1
  Inbound inspection rule is not set
  Outgoing inspection rule is INSPECT
    ftp alert is on audit-trail is off timeout 3600
    http java-list 99 alert is off audit-trail is on timeout 3600
    dns alert is off audit-trail is off timeout 30
    tftp alert is off audit-trail is off timeout 30
    isakmp alert is off audit-trail is off timeout 30
    icmp alert is off audit-trail is off timeout 10
    udp alert is off audit-trail is off timeout 30
 inspection of router local traffic is enabled
  Inbound access list is INBOUND
  Outgoing access list is not set
```

To verify the user-defined port-mappings use the following commands:

```
Rack1R4#show ip port-map | inc http
Default mapping:  http                 tcp port 80
system defined
Default mapping:  http                 tcp port 8080
user defined
Default mapping:  https                tcp port 443
system defined

Rack1R4#show ip port-map | inc ftp
Default mapping:  ftps                 tcp port 990
system defined
Default mapping:  tftp                 udp port 69
system defined
Default mapping:  ftp                  tcp port 21
system defined
Host specific:    ftp                  tcp port 80             in list
98  user defined
```

11.18 CBAC TCP/UDP Intercept Feature

- Configure R6 to protect the servers on VLAN 146 subnet from TCP/UDP based DoS attacks per the requirements below.
- Allow for a maximum of 100 half-open connections and configure the firewall to keep dropping the sessions until their number reaches 80.
- Limit the new connections rate to 30 per minute and clamp it down to 15 connections per second once the threshold has been crossed.
- Limit the maximum number of half-open TCP connections per server to 10. Block any new connections for 60 seconds when the number exceeds this limit.

Configuration

```
R6:
ip inspect max-incomplete low 80
ip inspect max-incomplete high 100
ip inspect one-minute low 15
ip inspect one-minute high 30
ip inspect tcp max-incomplete host 10 block-time 1
ip inspect tcp synwait-time 10
!
ip inspect name DEFEND tcp
ip inspect name DEFEND udp
!
interface FastEthernet 0/0.146
 ip inspect DEFEND out
```

Verification

Note

By default, with every inspection rule, CBAC enables DoS prevention features very similar to the TCP Intercept feature. However, with CBAC you cannot disable this behavior (only in some recent IOS releases, since 12.4(10) and later 12.4T trains). Another difference is that CBAC applies DoS prevention to UDP sessions as well. A UDP session is considered half-open if no packets have been sent in response to the initial packet. Both UDP and TCP half-open sessions count against maximum connection thresholds at the same time. Therefore, if you set up a limit of 100 half-open sessions the router will count both numbers of UDP and TCP sessions against this threshold.

CBAC supports maximum thresholds and per-minute rates for half-open sessions. When the current number of connections exceeds any of the thresholds, the code starts deleting half-open connection entries and sending RST packets to TCP session endpoints. The code does not delete any sessions that have already been established. For TCP sessions, the only supported mode of operations is “watch” mode. The code does not intervene with TCP sessions, just watches them as they get established. Offending sessions (sessions that did not complete in time or exceed thresholds) receive RST messages as well when they exceed configured timeouts. The amount of time CBAC waits for TCP connection establishment is configured using the command:

```
ip inspect tcp synwait-time <N>
```

Note this is the equivalent of “watch-timeout” for the TCP Intercept feature.

One feature special to CBAC is per-host TCP connection limits. CBAC bounds the number of half-open connections per-host, in addition to having the global thresholds. As the number of half-open sessions destined to the same IP address exceeds the per-host threshold, the router can block any further connections from establishing for the amount of time specified. The command syntax is:

```
ip inspect tcp max-incomplete host <N> block-time <T>
```

Lastly, remember that the CBAC interception feature applies whether you configure any inspection rule. In our case, the inspection rule simply inspects any TCP or any UDP traffic.

To verify how CBAC works, configure R6 to drop half-open sessions if they do not establish in 1 second.

```
R6:
ip inspect tcp synwait-time 1

Rack1R4#show ip inspect all
Rack1SW1#debug ip tcp transactions
TCP special event debugging is on

Rack1SW1#telnet 155.1.146.44
Trying 155.1.146.44 ...
TCB03A09440 created
TCB03A09440 setting property TCP_VRFTABLEID (14) 3A914FE
TCB03A09440 setting property TCP_TOS (1) 3A91458
TCB03A09440 bound to 0.0.0.0.11015
TCP: sending SYN, seq 3329226702, ack 0
TCP0: Connection to 155.1.146.44:23, advertising MSS 536
TCP0: state was CLOSED -> SYNSENT [11015 -> 155.1.146.44(23)]
TCP0: bad seg from 155.1.146.44 -- Rst bit set.: port 11015 seq 0 ack 0
rcvnxt 0 rcvwnd 4128 len 0
TCP0: timeout #1 - timeout is 4000 ms, seq 3329226702
TCP0: bad seg from 155.1.146.44 -- Rst bit set.: port 11015 seq 0 ack 0
rcvnxt 0 rcvwnd 4128 len 0
TCP0: timeout #2 - timeout is 8000 ms, seq 3329226702
TCP0: bad seg from 155.1.146.44 -- Rst bit set.: port 11015 seq 0 ack 0
rcvnxt 0 rcvwnd 4128 len 0
TCP0: timeout #3 - timeout is 15991 ms, seq 3329226702
TCP0: bad seg from 155.1.146.44 -- Rst bit set.: port 11015 seq 0 ack 0
rcvnxt 0 rcvwnd 4128 len 0
% Connection timed out; remote host not responding
```

Note the RST packets sent from the IP address “155.1.146.44”. Those are packets actually generated by R6 to terminate connections initiated from SW1.

11.19 VLAN Filtering for IP Traffic

- Restrict traffic on VLAN 146 so that only R1, R4, and R6 are allowed to communicate.
- Allow RIP routing updates as well as transit TCP connections on VLAN 146.
- Disallow any other traffic to cross VLAN 146.

Configuration

```
SW1:
!
! The following filter denies any IP traffic
! that does not match the access-list in the first entry
! All Layer 2 traffic is permitted by default
!
ip access-list extended ALLOWED_L3_TRAFFIC
 permit udp any any eq 520
 permit tcp any any
!
vlan access-map VLAN146_FILTER 10
 match ip address ALLOWED_L3_TRAFFIC
 action forward
!
! Apply the filter to the VLAN
!
vlan filter VLAN146_FILTER vlan-list 146
```

Verification

Note

VLAN filters on the Catalyst switches apply to any traffic ingress on the VLAN they attach to. The concept of a VLAN filter is an extension of access-lists to a whole bridged domain. It is important to remember the following key features of VLAN filters:

1) VLAN filters are ingress and thus conflict with any other ingress filtering features, such as port access-lists, either IP or MAC-based, and ingress SVI access-lists. You can still use outbound IP access-lists on SVIs.

2) VLAN filters are organized as a sequence of entries. Each entry matches either an IP or MAC access-list and you can specify an action, which is either forward or drop. Optionally, you can omit matching any access-list and then the entry would matches all IP and non-IP traffic.

If a packet does not match the entry, the next vlan-filter entry in sequence is attempted. Note that if a packet is denied in the access-list, it does not automatically mean it is going to be dropped. It just does not match the particular VLAN filter entry.

3) VLAN filters distinguish between IP and non-IP traffic based on the access-list matched. By default, a VLAN filter permits both types of traffic, until you match an access-list in a vlan-filter entry. If there is a particular type of access-list in a vlan-filter (e.g. IP access-list) then all non-matching traffic of this type is dropped by the filter. The other type of traffic (e.g. non-IP) can still pass the filter if there are no explicit access-lists of this type.

4) VLAN filters apply to both local traffic (inside the VLAN) and transit traffic. Also remember to re-apply your vlan-filter once you have changed its configuration since those filters are programmed in hardware.

This configuration can be verified as follows:

```
Rack1R4#ping 155.1.146.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

ICMP traffic from R4 to R1 is dropped as it enters SW1.

```
Rack1R4#telnet 155.1.146.1
Trying 155.1.146.1 ... Open


User Access Verification

Password:
Rack1R1>
```

TCP traffic, such as the above telnet session, is still allowed.

11.20 VLAN Filters for Non-IP Traffic

- Allow only the following Layer 2 protocols across VLAN 146
  - STP BPDUs (consider both ISL and 802.1q trunks)
  - CDP, VTP, and UDLD
  - ARP
- Disallow any other Layer 2 packets.

Configuration

```
SW1:
!
! The list below matches STP and ARP packets on the VLAN
!
mac access-list extended ALLOWED_L2_TRAFFIC
 permit any any lsap 0x4242 0x0
 permit any any 0x010B 0x0
 permit any any 0x806 0x0
!
! The list is then applied using the vlan filter
!
vlan access-map VLAN146_FILTER 20
 match mac address ALLOWED_L2_TRAFFIC
 action forward
!
! Apply the filter to the VLAN
!
vlan filter VLAN146_FILTER vlan-list 146
```

Verification

Note

When you perform Layer 2 filtering, you usually do that based on Ethertypes or LSAP (Link Service Access Point) values. LSAP value consists of two parts (one byte each): SSAP and DSAP (Source SAP) and (Destination SAP). You can apply Layer 2 filtering based on MAC addresses, but this is not very useful since MAC access-lists only match non-IP traffic (e.g. ARP, STP, VTP). A list of the most commonly seen Layer 2 protocols is outlined below:

IEEE STP BPDUs: These BPDUs use 802.2 LLC encapsulation with SSAP/DSSP values of 0x42 or LSAP value of 0x4242. Cisco switches and routers run IEEE STP over access-ports and run it across the native VLAN of a 802.1q trunk. You can also see STP packets sent across ISL trunks using the same LSAP value of 0x42. The Cisco implementation does not modify the STP packets sent over ISL trunks, it just adds the ISL encapsulation.

PVST+ SSTP BPDUs: These BPDUs use 802.2 SNAP encapsulation (LSAP=0xAAAA) with SNAP PID (Protocol ID) value of 0x010B. You can match this Protocol ID value as the Ethertype number in MAC ACLs in the Catalyst Switches. Cisco switches send PVST+ BPDUs across 802.1q only trunks in addition to IEEE STP BPDUs. The PVST+ BPDUs appear on native as well as non-native VLANs of a trunk port.

ARP protocol: This uses an Ethernet II frame format with the Ethertype value of 0x806

The protocols below use 802.2 SNAP encapsulation with the SNAP Protocol ID values listed below:

VTP: 0x2003
CDP: 0x2000
DTP: 0x2004
UDLD: 0x0111

All SNAP-encapsulated packets can be matched using an LSAP value of 0xAAAA. The above mentioned packet types have no VLAN tag header, so you can filter them on the native VLAN of a trunk, which is usually VLAN 1.

☠ Pitfall

Be very careful when filtering Layer 2 protocols as you can block STP BPDUs and effectively introduce topology loops and broadcast traffic storms.

✐ Note

For verification, make sure that you can see SW1 correctly executing the STP protocol. To accomplish this, configure SW2 as a STP root bridge and make sure that SW1 blocks redundant uplink ports.

```
SW2:
spanning-tree vlan 146 root primary
```

```
Rack1SW1#show spanning-tree vlan 146

VLAN0146
  Spanning tree enabled protocol ieee
  Root ID    Priority    24722
             Address     0019.55af.7100
             Cost        19
             Port        15 (FastEthernet0/13)
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID  Priority    32914  (priority 32768 sys-id-ext 146)
             Address     001a.a174.1e00
             Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
             Aging Time 15

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------
Fa0/1            Desg FWD 19        128.3    P2p
Fa0/13           Root LRN 19        128.15   P2p
Fa0/14           Altn BLK 19        128.16   P2p
Fa0/15           Altn BLK 19        128.17   P2p
Fa0/16           Altn BLK 19        128.18   P2p
Fa0/17           Altn BLK 19        128.19   P2p
Fa0/18           Altn BLK 19        128.20   P2p

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------

Fa0/19           Altn BLK 19        128.21   P2p
Fa0/20           Altn BLK 19        128.22   P2p
Fa0/21           Altn BLK 19        128.23   P2p
```

Now try running some protocol that the Catalyst switches treat as non-IP, e.g. IPX. Configure IPX on R1 and R4 and try pinging R1's IPX addresses from R4:

```
R1:
ipx routing
!
interface FastEthernet 0/0
 ipx network 146 encapsulation snap

R4:
ipx routing
!
interface FastEthernet 0/1
 ipx network 146 encapsulation snap

Rack1R1#show ipx interface fastEthernet 0/0
FastEthernet0/0 is up, line protocol is up
  IPX address is 146.000d.edc8.4f60, SNAP [up]
  Delay of this IPX network, in ticks is 1
```

```
Rack1R4#ping 146.000d.edc8.4f60

Translating "146.000d.edc8.4f60"

Type escape sequence to abort.
Sending 5, 100-byte IPX Novell Echoes to 146.000d.edc8.4f60, timeout is
2 seconds:
.....
Success rate is 0 percent (0/5)
```

The IPX packets are filtered as they arrive into VLAN 146 on SW1. Now remove the filter on SW1 and try pinging again.

```
Rack1SW1#config t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1SW1(config)#no vlan filter VLAN146_FILTER vlan-list 146
Rack1SW1(config)#

Rack1R4#ping 146.000d.edc8.4f60

Translating "146.000d.edc8.4f60"

Type escape sequence to abort.
Sending 5, 100-byte IPX Novell Echoes to 146.000d.edc8.4f60, timeout is
2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

11.21 Port Security

- Configure the respective switches to guard VLAN146 from MAC address flooding attacks originated at R1, R4, or R6.
- Limit the number of MAC addresses learned simultaneously on a single port to one.
- In case of a policy violation, apply the shutdown action on the port connected to R1 but recover the port after 3 minutes.
- For the port connected to R4, drop offending packets and generate log records of the violation.
- For the port connected to R6, simply drop the offending traffic.
- Age the learned secure entries after 10 minutes of inactivity.
- Retain the MAC addresses learned on the port connected to R4 in the switch configuration.

Configuration

```
SW1:
!
! R1 will be shut down in case of violation
!
interface FastEthernet0/1
 switchport mode access
 switchport port-security
 switchport port-security aging time 10
 switchport port-security aging type inactivity
!
! Recover the port from shutdown after 3 minutes
!
errdisable recovery cause psecure-violation
errdisable recovery interval 180

SW2:
!
! Protect the switch from port-security violations
! R6 connects via a trunk port so we specify the total amount
! plus the per-VLAN limits
!
interface FastEthernet0/6
 switchport mode trunk
 switchport port-security
 switchport port-security aging time 10
 switchport port-security aging type inactivity
 switchport port-security violation protect
 switchport port-security maximum 1 vlan 146
 switchport port-security maximum 1 vlan 67
 switchport port-security maximum 2
```

```
SW4:
!
! Learn tha MAC address of R4 as sticky
! Generate logging messages from violating packets
!
interface FastEthernet0/4
 switchport mode access
 switchport port-security
 switchport port-security aging time 10
 switchport port-security aging type inactivity
 switchport port-security violation restrict
 switchport port-security mac-address sticky
```

Verification

✐ Note

Port Security is a Layer 2 feature that enforces a limit on the number of MAC addresses allowed per a particular switch port. The two main purposes of this feature is preventing unauthorized connections on a shared port and thwarting MAC-address flooding attacks. The MAC address flooding attack consists of sending a barrage of packets with different MAC addresses forcing the switch to overpopulate its MAC address table. This attack may result in switch performance degradation and privacy violation. The latter may occur in cases when the switch starts behaving like a hub, flooding frames out all ports. This occurs when the MAC address table overflows.

Note that port-security works only on ports configured as static access or static trunks. The port-security feature does not work on dynamic ports. Port-security logic is outlined below.

1) On a port configured for port-security, the switch keeps a table of secure MAC address entries. The total number of entries allowed in this table is configurable using the following command:

```
switchport port-security maximum-address <N>
```

On trunk ports, the above command specifies the maximum number of MAC addresses active on all VLANs at the same time – the aggregate limit. Note that the switch treats the same MAC address on different VLANs as two different MAC addresses. For trunk ports, you may additionally specify the maximum number of MACs per VLAN using the command:

```
switchport port-security maximum <N> vlan <VLAN>
```

If the port is an access-port configured with access and voice VLANs, you can use commands to impose restrictions on just two VLANs, without ever mentioning their numbers:

```
switchport port-security maximum <N> vlan {access|voice}
```

When a switch has learned all allowed addresses on the port or a VLAN and a frame with a new source MAC address arrives on the port, the switch may take any of the following actions:

a) Shutdown the port (the default “shutdown” action).

b) Silently discard the frame (if configured for the “protect” action). Protect mode is not recommended for trunks ports, although we use it in this task. The problem is that as soon as any VLAN on a trunk reaches its MAC address limit, the port stops learning MAC addresses on any other VLAN. The worst thing about this mode is that the switch does not notify you about this via any logging message.

c) Discard the frame and generate a syslog message or SNMP trap (if configured for the “restrict” action). You may need to configure SNMP destination hosts to send the actual traps.

2) The switch does not allow the same MAC address to appear on more than one secure port at the same time. Thus, if a switch has learned a MAC address on a secure port it will not allow the same address to appear on other switch ports until the secure entry has expired.

3) The switch ages out secure MAC address entries using a configurable timeout. You can set this timeout per-port using two commands:

```
switchport port-security aging timeout <TIMEOUT>
switchport port-security aging type {absolute|inactivity}
```

Together these commands define the aging behavior. Absolute aging means age each entry starting at the time it has been learned. Inactivity aging starts aging time only after each frame is received. If no frames have been received during the timeout, the switch removes the secure entry and opens a slot for a different MAC address.

If the port-security feature has shut down a port, the port can be restored to an operational state using the error-disable recovery procedure. You can set one global recovery timeout (used for all types of error-disable recovery) by using the command:

```
errdisable recovery interval <seconds>
```

In order to enable a particular type of recovery procedure use the following command:

```
errdisable recovery cause <cause>
```

In addition to setting up dynamic learning of secure MAC addresses, you may configure static secure MAC address entries using the interface-level command:

```
switchport port-security mac-address H.H.H
```

These entries also count against the maximum number of allowed MAC addresses on an interface. You may configure a port to age static secure MAC address entries as well using the command:

```
switchport port-security aging static
```

This may be useful when you need to set up guaranteed access for a specific MAC address for some amount of time.

The last port-security feature is known as sticky learning. It allows you to learn "static" secure MAC address entries. When a switch learns new MAC addresses on a port in sticky mode, it generates a configuration line for a corresponding static entry. This line appears in the running configuration so you need to save it to make the static entry truly permanent.

For verification in our task, display the port-security configuration for all ports configured:

```
Rack1SW1#show port-security interface fastEthernet 0/1
Port Security              : Enabled
Port Status                : Secure-up
Violation Mode             : Shutdown
Aging Time                 : 10 mins
Aging Type                 : Inactivity
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 1
Total MAC Addresses        : 1
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 0
Last Source Address:Vlan   : 000c.31ef.4e60:146
Security Violation Count   : 0

Rack1SW2#show port-security interface fastEthernet 0/6
Rack1SW2#show port-security interface fastEthernet 0/6
Port Security              : Enabled
Port Status                : Secure-up
Violation Mode             : Protect
Aging Time                 : 10 mins
Aging Type                 : Inactivity
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 2
Total MAC Addresses        : 2
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 0
Last Source Address:Vlan   : 0011.200a.f000:146
Security Violation Count   : 0

Rack1SW2#show port-security interface fastEthernet 0/6 vlan 146
Default maximum: not set, using 6272
VLAN  Maximum    Current
  146          1          1

Rack1SW2#show port-security interface fastEthernet 0/6 vlan 67
Default maximum: not set, using 6272
VLAN  Maximum    Current
   67          1          1

Rack1SW4#show port-security interface fastEthernet 0/4
Port Security              : Enabled
Port Status                : Secure-up
Violation Mode             : Restrict
Aging Time                 : 10 mins
Aging Type                 : Inactivity
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 1
Total MAC Addresses        : 1
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 1
Last Source Address:Vlan   : 000f.23d5.2e81:146
Security Violation Count   : 0
```

Verify the sticky MAC address configuration:

```
Rack1SW4#show running-config interface fastEthernet 0/4
Building configuration...

Current configuration : 349 bytes
!
interface FastEthernet0/4
 switchport access vlan 146
 switchport mode access
 switchport port-security
 switchport port-security aging time 10
 switchport port-security violation restrict
 switchport port-security aging type inactivity
 switchport port-security mac-address sticky
 switchport port-security mac-address sticky 000f.23d5.2e81
```

Configure R4 to violate port-security. For this to happen, change the MAC address of the port on R4 connected to SW4. This will prevent R4 from communicating with other hosts until the old secure entry expires.

```
Rack1R4#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R4(config)#interface fastEthernet 0/1
Rack1R4(config-if)#mac-address 000f.23d5.2e82

Rack1SW4#
%PORT_SECURITY-2-PSECURE_VIOLATION: Security violation occurred, caused by MAC
address 000f.23d5.2e82 on port FastEthernet0/4.
%PORT_SECURITY-2-PSECURE_VIOLATION: Security violation occurred, caused by MAC
address 000f.23d5.2e82 on port FastEthernet0/4.
```

11.22 HSRP and Port-Security

- Reconfigure R3 and R4 temporarily for this task.
- Configure R1 and R3 to emulate an HSRP virtual router on VLAN 146 with the IP address 155.1.146.254.
- Configure the R4's Fa0/1 interface and the R6's VLAN 146 interface to emulate a virtual HSRP router on VLAN 146 with the IP address 155.1.146.253.
- Ensure your new configuration works with the port-security enabled ports.
- Do not use the **`switchport port-security maximum`** command on the port connected to R4.

Configuration

```
R1:
interface FastEthernet 0/0
 standby 1 ip 155.1.146.254

R3:
interface FastEthernet 0/1
 standby 1 ip
 no shutdown

R4:
!
! Temporarily shutdown R4's VLAN146 interface
!
interface FastEthernet 0/1
 shutdown
!
! We use a different group and IP for R4 and R6
! as they share the same VLAN as R1 and R3
!
! Allow R4 to use burned in MAC address for HSRP
!
interface FastEthernet 0/0
 ip address 155.1.146.4 255.255.255.0
 standby 2 ip
 standby use-bia

R6:
!
! Allow R6 to use burned in MAC address for HSRP
!
interface FastEthernet 0/0.146
 standby 2 ip 155.1.146.253
 standby use-bia
```

```
SW1:
!
! Allow R1 to use a virtual MAC address
!
interface FastEthernet 0/1
 switchport port-security maximum 2
!
! Temporarily reconfigure R3’s connection
!
interface FastEthernet 0/3
 switchport access vlan 146
 switchport mode access

SW2:
!
! Temporarily reconfigure R4’s connection
!
interface FastEthernet 0/4
 switchport access vlan 146
 switchport mode access
```

Verification

Note

Virtual router protocols (HSRP, VRRP and GLBP) use virtual MAC addresses in addition to physical interface addresses. This requires small changes in port-security configurations. At the very least, you need to increase the maximum MAC address count to permit additional entries in the secure MAC address table. HSRP offers another option allowing you to use only physical MAC-addresses, even for the virtual IP address.

First look at the output of the switch show commands and logging buffer contents before you changed the MAC address limit on SW1. Also, check the HSRP status on R4 before you enable the use of a BIA on R6:

```
Rack1SW1#
%PORT_SECURITY-2-PSECURE_VIOLATION: Security violation occurred, caused
by MAC address 0000.0c07.ac01 on port FastEthernet0/1.
```

```
Rack1SW1#show port-security interface fastEthernet 0/1
Port Security              : Enabled
Port Status                : Secure-shutdown
Violation Mode             : Shutdown
Aging Time                 : 10 mins
Aging Type                 : Inactivity
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 1
Total MAC Addresses        : 0
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 0
Last Source Address:Vlan   : 0000.0c07.ac01:146
Security Violation Count   : 1

Rack1R4#show standby
FastEthernet0/0 - Group 2
  State is Learn
  Virtual IP address is unknown
  Active virtual MAC address is unknown
    Local virtual MAC address is 0000.0c07.ac02 (v1 default)
  Hello time 3 sec, hold time 10 sec
  Preemption disabled
  Active router is unknown
  Standby router is unknown
  Priority 100 (default 100)
  IP redundancy name is "hsrp-Fa0/0-2" (default)
```

Check the state of the R4 HSRP group once again and note the virtual MAC address, it is the address of R4:

```
Rack1R4#show standby
FastEthernet0/0 - Group 2
  State is Active
    2 state changes, last state change 00:13:50
  Virtual IP address is 155.1.146.253 (learnt)
  Active virtual MAC address is 000f.23d5.2e80
    Local virtual MAC address is 000f.23d5.2e80 (bia)
  Hello time 3 sec, hold time 10 sec
    Next hello sent in 0.893 secs
  Preemption disabled
  Active router is local
  Standby router is 155.1.146.6, priority 100 (expires in 7.857 sec)
  Priority 100 (default 100)
  IP redundancy name is "hsrp-Fa0/0-2" (default)
```

In addition, R1's port should recover by this time and show the total count of MAC addresses as two:

```
Rack1SW1#
%PM-4-ERR_RECOVER: Attempting to recover from psecure-violation err-
disable state on Fa0/1
%LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1,
changed state to up

Rack1SW1#show port-security interface fastEthernet 0/1
Port Security              : Enabled
Port Status                : Secure-up
Violation Mode             : Shutdown
Aging Time                 : 10 mins
Aging Type                 : Inactivity
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 2
Total MAC Addresses        : 2
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 0
Last Source Address:Vlan   : 000c.31ef.4e60:146
Security Violation Count   : 0
```

Revert all the changes we made so far for R4 and R3 to be part of VLAN 146.

11.23 DHCP Snooping

- Configure R4 as a DHCP server and R1 as a DHCP client on VLAN 146.
- Configure SW1 to prevent potential DHCP attacks in such a way that it dynamically accounts for future hosts added to the VLAN 146 segment.
- SW1 should store the binding database in flash with the filename dhcp-bindings.txt, and use a 15 second delay between changes.
- Limit the amount of DHCP messages SW1 can receive from R1 to 10 per second.

Configuration

```
R1:
interface FastEthernet 0/0
 ip address dhcp

R4:
ip dhcp relay information trust-all
!
ip dhcp pool VLAN146
 network 155.1.146.0 /24

SW1:
ip dhcp snooping vlan 146
ip dhcp snooping
ip dhcp snooping database flash:/dhcp-bindings.txt
ip dhcp snooping database write-delay 15
!
interface FastEthernet 0/1
 ip dhcp snooping limit rate 10
!
! Trust DHCP on uplink that connect SW1 to R4
!
interface range FastEthernet 0/13 - 21
 ip dhcp snooping trust
```

Verification

Note

DHCP Snooping is a security feature that inspects DHCP packets transiting a Layer 2 switch. The switch itself does not participate in DHCP packet exchange but enforces DHCP packet flow integrity. Before we start with an overview of DHCP snooping, let us quickly recap different agent roles for IP address allocation procedures using DHCP.

1) A DHCP client discovers a DHCP server and requests an IP address as well as other configuration parameters using a DHCPDISCOVER packet. The client may then select an IP address using a DHCPREQUEST message. When the client no longer needs the IP address it returns it back to the server using a DHCPRELEASE packet.

2) A DHCP Relay acts as a middleman agent between the DHCP clients and the server. The primary function of a DHCP Relay is to forward messages from local clients to the remote DHCP Server. When a DHCP Relay receives a broadcast packet from a connected client it forwards it as a unicast packet to the IP address of the server. In the Cisco IOS, you can specify the IP address of the server using the interface-level command **`ip helper-address X.X.X.X`**. The DHCP relay changes the “giaddr” field in DHCP packets from zero to the IP address of the interface that forwards the packet. The DHCP server later uses this IP address to respond with a DHCP packet. A Catalyst switch may act as a DHCP Relay if you configure it as a Layer 3 device with an IP address and helper-address on some interface.

3) A DHCP Server responds to discover and request messages from DHCP clients with DHCPOFFER and DHCPACK/DHCPNAK messages. The former message contains the IP address for the client and the latter two acknowledge or reject a request message. The server uses the “giaddr” field to respond back provided that it is non-zero. A server may also explicitly request a client to release and IP address with a DHCPRELEASEREQUERY message.

Every DHCP message also carries the MAC address of the client explicitly stored in one of DHCP fields. This MAC address may identify a client to the server if the Client ID field is set in the packet.

The primary goal of using DHCP Snooping is to enforce DHCP security. When you enable DHCP Snooping on a switch, it starts treating every port as connected to a DHCP client (e.g. workstation, or some downstream switch). For such ports, also called “untrusted” ports, the switch applies DHCP message filtering, only accepting messages expected from DHCP Clients (DHCPREQUEST, DHCPDISCOVER, DHCPRELEASE).

The switch allows any type of DHCP messages on trusted ports. You may explicitly configure a port as trusted by the DHCP Snooping process using the interface-level command **`ip dhcp snooping trust`**. Usually, you need this command on the ports connected to DHCP servers or uplink ports on access switches.

On any type of port you can configure the command **`ip dhcp snooping limit rate <pps>`** to control the number of DHCP packets per second received on a particular port. If the port exceeds the threshold set, the switch will put the port into an error-disabled state. The port can be recovered from this state using common error-disable recovery procedures.

In addition, if an incoming DHCP message contains a non-zero “giaddr”, meaning it has been relayed, then it must be received on a trusted port. Untrusted ports simply discard such messages.

In addition to filtering messages based on their types, DHCP snooping also populates a special DHCP snooping table based on messages exchanged across untrusted ports. This table contains the IP addresses that were allocated to clients by the DHCP server and other information such as client MAC address, client port, VLAN number, and lease duration. The switch consults this table when it receives DHCP messages such as DHCPRELEASE to make sure that no one would spoof a DHCP message for another host. The switch also ensures that MAC addresses in a DHCP packet match the MAC address of the host sending the message. This feature could be disabled using the command **`no ip dhcp snooping verify mac-address`**. Note that messages received on trusted ports do not create any DHCP Snooping binding entries.

You may instruct the switch to save the DHCP snooping database contents in flash memory or on external server (e.g. via TFTP). This allows the switch to preserve DHCP binding mappings across reloads. The global command to specify the external storage is **`ip dhcp snooping database`**. You may also control the interval between database updates using the command **`ip dhcp snooping database write-interval <seconds>`**.

In order to start the DHCP Snooping process, you need to enable it globally as well as activate it per-VLAN. You need to do *both* in order for DHCP Snooping to start working. Note that by default, when DHCP Snooping is enabled, the switch also adds DHCP Information Option (or Option 82) to all received packets. The purpose of this option is to identify the device and port that the client connects to. In short, the option contains the ID of the switch (e.g. MAC address or hostname) that inserted this option and the port identifier. This can be an interface name that the DHCP Client connects to. This information allows the DHCP Server to allocate IP addresses based on additional information, for example, split a subnet between two port ranges in the switch that inserted the option.

One issue that may arise here is that the switch inserts the option but leaves the “giaddr” field at zero. Thus, a DHCP Server may assume that option has been formatted incorrectly, since a DHCP Relay is supposed to set the “giaddr” field to its own IP address. Thus, an IOS DHCP server will reject by default such DHCP messages. To overcome this issue you may use either of the following methods:

1) Instruct the IOS DHCP Server to accept DHCP messages with a zero “giaddr” using the global command: **`ip dhcp relay information trust-all`** or the interface-level command **`ip dhcp relay information trusted`**.

2) Configure the DHCP Snooping feature in the switch not to insert Option 82. This is accomplished using the command: **`no ip dhcp-snooping information option`**

3) Trust the port where you receive the original DHCP message. The DHCP Snooping feature does not insert any Information Option into the received packets.

Here we verify the status of DHCP snooping on SW1:

```
Rack1SW1#show ip dhcp snooping
Switch DHCP snooping is enabled
DHCP snooping is configured on following VLANs:
146
Insertion of option 82 is enabled
   circuit-id format: vlan-mod-port
    remote-id format: MAC
Option 82 on untrusted port is not allowed
Verification of hwaddr field is enabled
Interface                    Trusted     Rate limit (pps)
------------------------     -------     ----------------
FastEthernet0/1              no          10
FastEthernet0/13             yes         unlimited
FastEthernet0/14             yes         unlimited
FastEthernet0/15             yes         unlimited
FastEthernet0/16             yes         unlimited
FastEthernet0/17             yes         unlimited
FastEthernet0/18             yes         unlimited
FastEthernet0/19             yes         unlimited
FastEthernet0/20             yes         unlimited
FastEthernet0/21             yes         unlimited
```

```
Rack1SW1#show ip dhcp snooping binding
MacAddress          IpAddress        Lease(sec)  Type           VLAN
Interface
------------------  ---------------  ----------  -------------  ----
00:0C:31:EF:4E:60   155.1.146.5      85188       dhcp-snooping  146
FastEthernet0/1
Total number of bindings: 1
```

Note the limits set for the port connected to R1. You can also check the DHCP Snooping database contents using the command below:

```
Rack1SW1#show ip dhcp snooping database detail
Agent URL : flash:/dhcp-bindings.txt
Write delay Timer : 15 seconds
Abort Timer : 300 seconds

Agent Running : No
Delay Timer Expiry : Not Running
Abort Timer Expiry : Not Running

Last Succeded Time : 00:31:44 UTC Mon Mar 1 1993
Last Failed Time : None
Last Failed Reason : No failure recorded.

Total Attempts       :        2   Startup Failures :        0
Successful Transfers :        2   Failed Transfers :        0
Successful Reads     :        1   Failed Reads     :        0
Successful Writes    :        1   Failed Writes    :        0
Media Failures       :        0

First successful access: Read

Last ignored bindings counters :
Binding Collisions    :        0   Expired leases    :        0
Invalid interfaces    :        0   Unsupported vlans :        0
Parse failures        :        0
Last Ignored Time : None

Total ignored bindings counters:
Binding Collisions    :        0   Expired leases    :        0
Invalid interfaces    :        0   Unsupported vlans :        0
Parse failures        :        0
```

```
**Rack1SW1#more flash:/dhcp-bindings.txt**
2b915970
TYPE DHCP-SNOOPING
VERSION 1
BEGIN
155.1.146.5 146 000c.31ef.4e60 2B92AAB2 Fa0/1
4ba698f0
END
```

The last command shows the raw database contents, as they are stored in flash memory.

11.24 DHCP Snooping and the Information Option

- Configure R6 to obtain IP address information via DHCP from R4.
- Ensure that SW2 inserts Option 82 in DHCP requests received from R6.
- SW2 should be configured so that it uses the string “SWITCH2” for Option 82 Remote-ID and the string “ROUTER6” as the Circuit-ID for the port connected to R6.
- SW2 should accept information options in DHCP packets even on non-trusted ports.

Configuration

```
R6:
interface FastEthernet 0/0.146
 ip address dhcp

SW2:
interface FastEthernet0/6
 ip dhcp snooping vlan 146 information option format-type circuit-id
string ROUTER6
!
ip dhcp snooping information option format remote-id string SWITCH2
ip dhcp snooping information option allow-untrusted
!
ip dhcp snooping
ip dhcp snooping vlan 146
!
interface range FastEthernet 0/13 - 18 , FastEthernet 0/21
 ip dhcp snooping trust
```

Verification

Note

DHCP Information Option (Option 82) is used in large Enterprise and Metro Ethernet deployments to enhance functionality of IP address allocation processes. The option itself has no strict format, rather the RFC defines a very flexible structure for a relay agent to insert any information in this option. Two commonly defined and used sub-fields are Remote-ID and Circuit-ID. The first field identifies the remote device that inserted the option – that is, the DHCP relay that redirected the original request. The second field identifies the point of client’s attachment, which is commonly a port name or some other port identifier.

Over time, Cisco has been changing the format of Option 82 used in their switches. The most recent code uses the hostname as the default remote ID, but this can be changed to any ASCII string. For every port, you can also set a per-VLAN circuit ID as another ASCII string. This allows the use of different IDs for different VLANs on a trunk port.

The switch does not accept DHCP packets with a non-zero "giaddr" on untrusted ports. Remember, these are the ports facing DHCP clients. In addition, it does not accept DHCP packets with Option 82 on untrusted ports. However, the latter behavior can be modified using the command: **`ip dhcp snooping information option allow-untrusted`** which allows the switch to receive DHCP packets with Option 82 even on untrusted ports.

However, packets with a "giaddr" of zero are still rejected by the switch even with this command active in the configuration. You need to mark the respective port as "trusted" in order to accept such packets.

When you trust a port for DHCP Snooping operations, the switch does not create a snooping entry nor does it add any information option by itself for IP packets received on the trusted port.

For verification in our case, enable a DHCP packet content dump on R4 to read the Option 82 contents:

```
Rack1R4(config)#access-list 100 permit udp any any eq bootps

Rack1R4#debug ip packet 100 dump

IP: s=0.0.0.0 (FastEthernet0/1), d=255.255.255.255, len 352, rcvd 2
07E3C1A0:                         FFFF FFFFFFFF           ......
07E3C1B0: 0011200A F0000800 45000160 87D00000  .. .p...E..`.P..
07E3C1C0: FE1133BD 00000000 FFFFFFFF 00440043  ~.3=.........D.C
07E3C1D0: 014CBE6D 01010600 00001907 00008000  .L>m............
07E3C1E0: 00000000 00000000 00000000 00000000  ................
07E3C1F0: 0011200A F0000000 00000000 00000000  .. .p...........
07E3C200: 00000000 00000000 00000000 00000000  ................
07E3C210: 00000000 00000000 00000000 00000000  ................
07E3C220: 00000000 00000000 00000000 00000000  ................
07E3C230: 00000000 00000000 00000000 00000000  ................
07E3C240: 00000000 00000000 00000000 00000000  ................
07E3C250: 00000000 00000000 00000000 00000000  ................
07E3C260: 00000000 00000000 00000000 00000000  ................
07E3C270: 00000000 00000000 00000000 00000000  ................
07E3C280: 00000000 00000000 00000000 00000000  ................
07E3C290: 00000000 00000000 00000000 00000000  ................
07E3C2A0: 00000000 00000000 00000000 00000000  ................
07E3C2B0: 00000000 00000000 00000000 00000000  ................
07E3C2C0: 63825363 35010139 0204803D 1F006369  c.Sc5..9...=..ci
07E3C2D0: 73636F2D 30303131 2E323030 612E6630  sco-0011.200a.f0
07E3C2E0: 30302D46 61302F30 2E313436 0C075261  00-Fa0/0.146..Ra
```

```
07E3C2F0: 636B3152 36370801 060F2C03 21962B52  ck1R67....,.!.+R
07E3C300: 16010901 07524F55 54455236 02090107  .....ROUTER6....
07E3C310: 53574954 434832FF                    SWITCH2.
```

Note the Client-ID that contains the name of the VLAN 146 interface of R6 and the Option 82 contents containing the two strings configured in SW2.

Now try simulating a DHCP request with a non-zero “giaddr” field across SW2. To accomplish this, configure a new DHCP pool in R4 and request a DHCP address for SW3 on VLAN 79:

```
R4:
ip dhcp pool VLAN79
 network 155.1.79.0 /24

SW1:
interface Vlan79
 ip helper-address 155.1.146.4

SW3:
interface Vlan 79
 ip address dhcp

Rack1SW2#debug ip dhcp snooping event
DHCP Snooping Event debugging is on

DHCP_SNOOPING: received new DHCP packet from input interface
(FastEthernet0/6)
DHCP_SNOOPING: process new DHCP packet, message type: DHCPDISCOVER,
input interface: Fa0/6, MAC da: 0015.c634.6c61, MAC sa: 000c.ce65.50a0,
IP da: 155.1.146.4, IP sa: 155.1.79.7, DHCP ciaddr: 0.0.0.0, DHCP
yiaddr: 0.0.0.0, DHCP siaddr: 0.0.0.0, DHCP giaddr: 155.1.79.7, DHCP
chaddr: 000f.f76d.ac80
%DHCP_SNOOPING-5-DHCP_SNOOPING_NONZERO_GIADDR: DHCP_SNOOPING drop
message with non-zero giaddr or option82 value on untrusted port,
message type: DHCPDISCOVER, MAC sa: 000c.ce65.50a0
```

As the debugging output reveals, the switch does not accept those packets on untrusted ports. It would still accept the packets with a zero “giaddr”. These are the packets with a DHCP option inserted by some other Layer 2 device.

11.25 Dynamic ARP Inspection

- Configure SW2 to prevent ARP poisoning attacks on VLAN 146.
- Do not trust any trunk ports to other switches, but ensure that the switch enforces ARP security for R4 and R1.
- Log all ARP packets permitted by the ARP access-list but limit their rate to four per ten seconds.
- Additionally, log all packets matched against the DHCP snooping database.
- Store, at maximum, 16 entries in the ARP logging buffer.
- Enable all additional sanity checks for ARP packets.

Configuration

```
SW2:
!
! ARP Access-List
!
arp access-list ARP_VLAN146
 permit ip host 155.1.146.1 mac host 000d.edc8.4f60 log
 permit ip host 155.1.146.4 mac host 0015.c634.6c61 log
!
! Apply ARP Inspection
!
ip arp inspection vlan 146
ip arp inspection vlan 146 logging acl-match matchlog
ip arp inspection log-buffer entries 16
ip arp inspection log-buffer logs 4 interval 10
ip arp inspection validate src-mac dst-mac ip
!
! Apply the ARP ACL
!
ip arp inspection filter ARP_VLAN146 vlan  146
```

Verification

Note

Dynamic ARP inspection is a security feature that fixes some well-known weaknesses of the ARP protocol. Generally, ARP operates on a broadcast Ethernet segment, and allows any host to spoof a MAC address for any IP address on the segment. These attacks, commonly known as “Man-in-the-Middle” (MiM) attacks, cannot be prevented by using just port-security, access-lists, or other well-known features.

ARP Inspection creates a special IP to MAC address binding table in the switch. This table is dynamically populated based on the DHCP snooping database contents. You can also add static entries to the database manually, using the configuration commands for ARP Inspection access-lists.

When the switch receives an ARP packet on an ARP-untrusted (the default state) port it inspects the packet contents. Based on the IP to MAC address binding information in the packet, the switch permits the packet only if it matches the ARP Inspection table. This prevents ARP poisoning attacks.

Note that implementing ARP Inspection may break some services, such as Proxy ARP. To resolve these issues, ARP Inspection allows you to configure some ports as trusted for ARP Inspection. On trusted ports, the switch does not inspect any ARP message. It is common to trust ARP messages on switch uplink ports, pointing toward the network core. The command to make a port trusted is **`ip arp inspection trust`**.

The ARP Inspection feature may perform additional checks on ARP packets. It can check that destination MAC addresses inside ARP packet bodies match the destination MAC address in Ethernet frames for ARP responses. In addition to this, the switch check source MAC addresses in ARP packets and Ethernet frames for ARP requests. You can enable these two checks using the commands

```
ip arp inspection validate src-mac
ip arp inspection validate dst-mac
```

Additionally, you may enable IP address consistency checks for ARP packets using the command **`ip arp inspection validate ip.`** This command checks source/destination IP addresses for consistency. Specifically, it ensures that no host tries to bind MAC addresses to IP addresses such as “255.255.255.255” or “0.0.0.0”.

When enabled by default, the IP ARP Inspection feature builds all ARP mapping information based on the DHCP bindings table. If there are hosts on the segment not using DHCP for address allocation, you need to configure ARP access-lists. The syntax to create and apply a filter is:

```
arp access-list <ACL_NAME>
 permit [request|response] ip <address> <mask> mac
<address> <mask> [log]

ip arp inspection filter <ACL_NAME> vlan <VLAN_ID> [static]
```

For the access-list, you specify an IP address and MAC address binding. Note that you commonly do not match ARP requests or responses and usually use just a host IP to host MAC address mapping. The ARP Inspection feature first checks the ARP access-list for a given VLAN to see if a given ARP packet is legitimate. If there are no permit matches found for the given IP and MAC address pair, and there is NO explicit **`deny ip any mac any`** statement in the end of the access-list, the feature also checks the DHCP bindings database. However, if there is an explicit deny statement in the end of access-list or the access-list has been applied with the **`static`** keyword, then ARP Inspection does not consult the DHCP Snooping database.

Logging of denied or allowed packets is somewhat complicated with ARP Inspection. First, you may specify a **`log`** keyword in the ARP access-list. Then you need to enable ARP inspection using the command:

```
ip arp inspection vlan <VLAN_ID> logging acl-match
{matchlog|none}
```

This will enable or disable (the default) logging of ARP packets matching ACL entries configured with the **`log`** keyword. In parallel with the above command, you may configure two additional commands:

```
ip arp inspection vlan <VLAN_ID> logging dhcp-bindings
{all|permit|none}
ip arp inspection vlan <VLAN_ID> logging arp-probe
```

Those two respective commands enable logging of packets matching the DHCP bindings database and the logging of special ARP probe packets (packets with a source IP of 0.0.0.0). By default, the switch logs ARP packets denied by the DHCP snooping database. You may change this mode by disabling the logging of DHCP denied packets or enable logging of permitted packets. You may even log all packets matched against the DHCP snooping entries.

The switch accumulates logged ARP packets in a special internal buffer. You can regulate the size of this buffer using the command:

```
ip arp inspection log-buffer entries <N>
```

The switch empties this buffer using a rate-limited procedure based on the following command settings:

```
ip arp inspection log-buffer logs <number> <interval>
```

Based on these settings, the switch will generate at maximum <*number*> of syslog messages during <*interval*> time. If you set the <*interval*> to zero, the switch will generate a message for every ARP packet to be logged.

The last security feature is ARP message rate-limiting. This feature is on by default on untrusted ports and disabled on trusted ports. On both types of ports you can enable this feature using the command:

ip arp inspection limit rate {<pps> [burst interval <seconds>]|none}

This command will restrict the rate of received ARP packets to <*pps*> per <*seconds*> interval. When the port exceeds this rate, the switch will bring it to the error-disable state. By default, the rate limit on untrusted ports is 15 pps. The feature limits the aggregate rate on trunks or EtherChannels, so you may need to adjust it in real-life situations.

For our tasks, verify that ARP is enabled for the particular VLAN:

```
Rack1SW2#show ip arp inspection vlan 146

Source Mac Validation      : Enabled
Destination Mac Validation : Enabled
IP Address Validation      : Enabled

 Vlan     Configuration    Operation   ACL Match          Static ACL
 ----     -------------    ---------   ---------          ----------
  146     Enabled          Active      ARP_VLAN146        No

 Vlan     ACL Logging      DHCP Logging      Probe Logging
 ----     -----------      ------------      -------------
  146     Acl-Match        Deny              Off
```

Note that ARP inspection is enabled on VLAN 146 and ARP logging for ACL entries is active. By default, DHCP Logging is also enabled.

Now clear the ARP cache on R6 and send ping packets to R1 and R4. These packets should be matched and logged by the ARP ACL.

```
Rack1R6#clear arp-cache
Rack1R6#ping 155.1.146.4

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.4, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms
```

```
Rack1R6#ping 155.1.146.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 m

Rack1SW2#show logging
<snip>

Log Buffer (4096 bytes):

%SW_DAI-6-ACL_PERMIT: 1 ARPs (Res) on Fa0/16, vlan
146.([000d.edc8.4f60/155.1.146.1/000c.ce65.50a0/155.1.146.2/14:27:51
UTC Fri Mar 5 1993])
%SW_DAI-6-ACL_PERMIT: 1 ARPs (Res) on Fa0/16, vlan
146.([0015.c634.6c61/155.1.146.4/000c.ce65.50a0/155.1.146.2/14:27:51
UTC Fri Mar 5 1993])
```

Now change the MAC address of the R6 VLAN 146 interface and observe how the switch denies the violating ARP packets:

```
Rack1R6#show interfaces fastEthernet 0/0
FastEthernet0/0 is up, line protocol is up
  Hardware is AmdFE, address is 000c.ce65.50a0 (bia 000c.ce65.50a0)
<snip>

Rack1R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R6(config)#interface fastEthernet 0/0
Rack1R6(config-if)#mac-address 000c.ce65.50a1

Rack1SW2#
%SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Fa0/6, vlan
146.([000c.ce65.50a1/155.1.146.2/0015.c634.6c61/155.1.146.4/14:32:51
UTC Fri Mar 5 1993])
%SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Fa0/6, vlan
146.([000c.ce65.50a1/155.1.146.2/000d.edc8.4f60/155.1.146.1/14:32:51
UTC Fri Mar 5 1993])
%SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Res) on Fa0/6, vlan
146.([000c.ce65.50a1/155.1.146.2/ffff.ffff.ffff/155.1.146.2/14:32:51
UTC Fri Mar 5 1993])
```

As you can see, there is no DHCP snooping entry to match the new R6 MAC address, so the ARP packets are dropped by the switch.

11.26 IP Source Guard

- Configure SW2 to prevent IP address spoofing on the port connected to R6.
- Ensure that your solution accounts for dynamic IP addresses obtained via DHCP.
- Enforce Layer 2 filtering for the MAC addresses corresponding to secured IP addresses at the same time.
- Do not shutdown ports in case of a security violation.

Configuration

```
SW2:
!
! Note that we set maximum allowed secure MAC addresses to 2
! since we have two VLANs on the trunk
!
interface FastEthernet 0/6
 ip verify source port-security
 switchport port-security maximum 2
 switchport port-security
!
! Enable DHCP snooping on both VLANs to ensure proper filtering
!
ip dhcp snooping
ip dhcp snooping vlan 67,146
!
! Since R6’s VLAN67 IP is static, we need a static IP/MAC/Port binding
! The MAC address here is R6’s physical MAC address
!
ip source binding 000c.ce65.50a0 vlan 67 155.1.67.6 interface
FastEthernet 0/6
```

Verification

Note

IP Source Guard is a feature intended to prevent IP packet spoofing at Layer 2 (MiM attacks). When you enable IP Source Guard on a port, the switch applies a Layer 3 filter to this port, only accepting the packets with source IP addresses matching DHCP snooping bindings created for the port. Enabling DHCP snooping is a prerequisite, but you may bind an IP address to a port manually, even if the host does not use DHCP.

As soon as you enable IP Source Guard, the switch only permits IP packets that match the DHCP snooping database or static IP to MAC addresses and port bindings. The switch also allows ingress DHCP packets for hosts to obtain IP addresses.

IP Source Guard relieves you from the need of applying any IP ingress filtering on individual ports to prevent IP address spoofing. Combined with Dynamic ARP Inspection and DHCP Snooping it allows much more secure environment than default Layer 2 deployments.

You may enable IP Source Guard per-interface using the command:

`ip verify-source [port-security]`

The **`port-security`** option requires port-security be enabled on the switch port as well. With this option, the switch filters packets both based on the source IP and MAC addresses, and the secure MAC address is taken from the DHCP snooping database or a static mapping entry.

Note that you may enable IP Source Guard on a trunk port as well. In this case, it requires DHCP snooping to be enabled on all trunked VLANs for filtering to work properly.

In our solution, we apply source guard to the ports connected to R6. Note that we need to enable DHCP snooping on both VLANs active on the trunk to ensure proper filtering. In addition, the port should be allowed to permit at least two MAC addresses, as there are two VLANs on the trunk.

Here we verify that IP Source Guard is active on the port:

```
Rack1SW2#show ip verify source
Interface  Filter-type  Filter-mode  IP-address       Mac-address        Vlan
---------  -----------  -----------  ---------------  -----------------  ------
Fa0/6      ip-mac       active       155.1.67.6       00:0C:CE:65:50:A0  67
Fa0/6      ip-mac       active       155.1.146.3      00:0C:CE:65:50:A0  146
```

Ensure filtering actually prevents IP address spoofing by changing the IP address of R6:

```
Rack1R6#ping 155.1.146.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms

Rack1R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R6(config)#interface fastEthernet 0/0.146
Rack1R6(config-subif)#ip address 155.1.146.254 255.255.255.0
Rack1R6(config-subif)#^Z
Rack1R6#ping 155.1.146.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
Rack1R6#

Rack1SW2#show ip verify source
Interface  Filter-type  Filter-mode  IP-address       Mac-address        Vlan
---------  -----------  -----------  ---------------  -----------------  ------
Fa0/6      ip-mac       active       155.1.67.6       00:0C:CE:65:50:A0  67
Fa0/6      ip-mac       active       deny-all         deny-all           146
```

Note the IP source guard entry for VLAN 146 has changed to deny all. Only DHCP packets are permitted in this situation, to allow the end static MAC to obtain an IP address.

11.27 Using Catalyst Ingress Access-Lists

- Allow R1 to only send ARP and ICMP packets out of its VLAN 146 connection.

Configuration

```
SW1:
!
! Allow ARP traffic only
!
mac access-list extended ARP_ONLY
 permit any any 0x806 0x0
!
! Allow ICMP traffic only
!
ip access-list extended ICMP_ONLY
 permit icmp any any
!
! Apply the access-list
!
interface FastEthernet0/1
 ip access-group ICMP_ONLY in
 mac access-group ARP_ONLY in
```

Verification

Note

You may apply IP or MAC access lists ingress on any Layer 2 port. You cannot apply them outbound on Layer 2 ports, only on SVI interfaces. In addition, the use of Layer 2 access-lists is limited to Layer 2 ports only.

Note one important difference. In the 3550 platform, MAC-based access-lists match only non-IP traffic such as ARP, STP, and IPv6. In the 3560 platform, they match only non-IP/IPv6 traffic such as ARP, STP, IPX and so on. You cannot filter IP traffic based on MAC address unless you use the port-security feature.

Be careful when applying MAC-based access-lists, you can filter management traffic such as STP BPDUs.

Verify that the access-list permits only ICMP traffic first:

```
Rack1R1#ping 155.1.146.4

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.146.4, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Rack1R1#telnet 155.1.146.4
Trying 155.1.146.4 ...
% Connection timed out; remote host not responding
```

Now check that IPX traffic is blocked:

```
R1:
ipx routing
!
interface FastEthernet 0/0
 ipx network 146 encapsulation snap

R4:
ipx routing
!
interface FastEthernet 0/1
 ipx network 146 encapsulation snap

Rack1R1#ping 146.0015.c634.6c61

Translating "146.0015.c634.6c61"

Type escape sequence to abort.
Sending 5, 100-byte IPX Novell Echoes to 146.0015.c634.6c61, timeout is
2 seconds:
.....
Success rate is 0 percent (0/5)
```

Remove the MAC access-list and check again that IPX traffic is now permitted:

```
Rack1SW1#interface FastEthernet 0/1
Rack1SW1(config-if)#no mac access-group ARP_ONLY in

Rack1R1#ping 146.0015.c634.6c61

Translating "146.0015.c634.6c61"

Type escape sequence to abort.
Sending 5, 100-byte IPX Novell Echoes to 146.0015.c634.6c61, timeout is
2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

11.28 Controlling Terminal Line Access

- Only allow Telnet connections to R2 from the Loopback0 subnets.
- Authenticate remote users locally using the name “TELNET” and the password of “CISCO”.
- Only allow this user to connect to R1 from R2.
- Log any attempt to connect from R2 to any destination on port 80.

Configuration

```
R2:
access-list 99 permit 150.1.0.0 0.0.255.255
!
access-list 100 permit ip any host 150.1.1.1
access-list 100 permit ip any host 155.1.146.1
access-list 100 permit ip any host 155.1.0.1
access-list 100 permit ip any host 155.1.13.1
access-list 100 deny tcp any any eq 80 log
!
username TELNET password CISCO
username TELNET access-class 100
!
line vty 0 4
 access-class 99 in
 login local
```

Verification

Note

You can apply standard or extended access-lists to any VTY line or username using the respective access-class. Based on the direction of the access-class command, it can control connections to the router (inbound), or from the router (outbound). When an access-list applies to a particular user, it controls the user’s outgoing connections from the router.

Using an extended access-list may be useful when you want to filter connections to a particular port. For example, when you want to deny connections to any rotary group port like 700X, or filter outgoing connections to port 80. When you use an extended access-list to match an incoming connection, use “any” as the destination IP address. The router will actually use the destination IP address 0.0.0.0 to match against the access-list, but using “any” is just a safeguard measure.

By specifying the **`log`** keyword, you may register packets matching a particular line in the access-list. This may be useful if you want to trace connections going to a particular port.

Try connecting to R2, sourcing the connection off the various interfaces:

```
Rack1R5#telnet 150.1.2.2
Trying 150.1.2.2 ...
% Connection refused by remote host

Rack1R5#telnet 150.1.2.2 /source Loopback0
Trying 150.1.2.2 ... Open


User Access Verification

Username: TELNET
Password: CISCO
```

Try connecting from R2 to different destinations and ensure that you can only connect to R1:

```
Rack1R2>telnet 150.1.3.3
Trying 150.1.3.3 ...
% Connections to that host not permitted from this terminal

Rack1R2>telnet 150.1.1.1
Trying 150.1.1.1 ... Open

User Access Verification

Password: cisco

Rack1R1>exit

[Connection to 150.1.1.1 closed by foreign host]
```

Now check that the router logs outgoing connections to port 80:

```
Rack1R2#terminal monitor
Rack1R2#telnet 150.1.3.3 80
Trying 150.1.3.3, 80 ...
% Connections to that host not permitted from this terminal
Rack1R2#
%SEC-6-IPACCESSLOGP: list 100 denied tcp 0.0.0.0(20034) ->
150.1.3.3(80), 1 packet
```

11.29 IOS Login Enhancements

- After 3 unsuccessful attempts to log into R3 in 30 seconds, block all further attempts for 40 seconds.
- Ensure that sessions sourced from the R5 Loopback0 are exempted from this restriction.
- Log every successful login attempt and every 3rd unsuccessful attempt.
- Enforce a delay of 2 seconds between successive login attempts.
- Create a local username “TEST” with the password of “TEST” for this task.

Configuration

```
R3:
username TEST password TEST
access-list 99 permit 150.1.5.5
!
login block-for 40 attempts 3 within 30
login quiet-mode access-class 99
login on-failure log every 3
login on-success log
login delay 2
!
line vty 0 4
 login local
```

Verification

Note

Login enhancements or IOS login block provides protection from brute-force dictionary attacks. This feature counts the number of failed login attempts during a time interval and prevents any further attempts for the duration of the block time interval. The command to set this behavior is:

`login block-for <seconds> attempts <tries> within <interval>`

This command will block any further login attempts for <*seconds*> if there were <*tries*> failed attempts during the <*interval*>. During this “quiet” interval, all login attempts are blocked with the exception of the hosts mentioned in the following access-list:

`login quiet-mode access-class {acl-name|acl-number}`

This is a special access-list which allows some important hosts to connect to the router even though it is blocking logins.

Additional commands for this feature include:

login on-failure log [every <login>]
login on-success log [every <login>]
login delay <seconds>

The first command enables logging every failed <*login*> attempt and the second command enables logging every unsuccessful <*login*> attempt. For example, if <*login*> is 3, then every 3rd attempt is logged. Without the parameter, these commands log every next attempt. The last command specifies the delay to enforce between every next login attempt to the router.

Note that these features are configured independent of any AAA settings. Additionally, banners and messages are also set separately from this feature.

Note that this feature does not work with line-based authentication, only with AAA or local database authentication.

Try logging in to R3 and fail authentication three times. Note that the router has logged the 3rd failed attempt.

```
Rack1R3#telnet 150.1.3.3
Trying 150.1.3.3 ... Open

User Access Verification

Username: TEST
Password:
% Login invalid

Username: TEST
Password:
% Login invalid

Username: TEST
Password:
% Login invalid

*Mar  6 12:23:49.920: %SEC_LOGIN-4-LOGIN_FAILED: Login failed [user:
TEST] [Source: 150.1.3.3] [localport: 23] [Reason: Login Authentication
Failed - BadPassword] at 12:23:49 UTC Wed Mar 6 2002
```

At the same time, the router will enter the quiet mode, blocking any further login attempts. After 40 seconds the quiet mode expires:

```
*Mar  6 12:23:49.920: %SEC_LOGIN-1-QUIET_MODE_ON: Still timeleft for
watching failures is 11 secs, [user: TEST] [Source: 150.1.3.3]
[localport: 23] [Reason: Login Authentication Failed - BadPassword]
[ACL: 99] at 12:23:49 UTC Wed Mar 6 2002
[Connection to 150.1.3.3 closed by foreign host]

Rack1R3#telnet 150.1.3.3
Trying 150.1.3.3 ...
% Connection refused by remote host

Rack1R3#
*Mar  6 12:24:29.922: %SEC_LOGIN-5-QUIET_MODE_OFF: Quiet Mode is OFF,
because block period timed out at 12:24:29 UTC Wed Mar 6 200
```

Now try logging in correctly and see if this attempt gets logged:

```
Rack1R3#telnet 150.1.3.3
Trying 150.1.3.3 ... Open


User Access Verification

Username: TEST
Password: TEST
Rack1R3>
*Mar  6 12:29:19.781: %SEC_LOGIN-5-LOGIN_SUCCESS: Login Success [user:
TEST] [Source: 150.1.3.3] [localport: 23] at 12:29:19 UTC Wed Mar 6
2002
```

Test the exemption for the quiet period. First try logging in three times incorrectly to R3 from R5:

```
Rack1R5#telnet 150.1.3.3
Trying 150.1.3.3 ... Open


User Access Verification

Username: TEST
Password:
% Login invalid

Username: TEST
Password:
% Login invalid

Username: TEST
Password:
% Login invalid

[Connection to 150.1.3.3 closed by foreign host]

Rack1R5#telnet 150.1.3.3
Trying 150.1.3.3 ...
% Connection refused by remote host

Rack1R5#telnet 150.1.3.3 /source Loopback0
Trying 150.1.3.3 ... Open


User Access Verification

Username: TEST
Password: TEST
Rack1R3>
```

While testing, note that the delay between each attempt is 2 seconds.

11.30 Role Based CLI

- Create roles in R4 named “SUPER”, “DEBUG”, “INTERFACE1” and “INTERFACE2”.
- The role “INTERFACE1” should have access to all IP configuration commands of interface Fa0/0 only. For “INTERFACE2” do the same with the exception that it applies to interface Fa0/1.
- The role “DEBUG” should be able to use any **debug** and **undebug** commands, and it should be able to inspect the running configuration.
- The last role name “SUPER” should be a sum of all the other roles.
- Use the password of “CISCO” to authenticate all roles.

Configuration

```
R4:
aaa new-model
!
exit
!
enable view
!
conf t
!
parser view DEBUG
 secret CISCO
 commands exec include show running-config
 commands exec include all debug
 commands exec include all undebug
!
parser view INTERFACE1
 secret CISCO
 commands interface include all ip
 commands configure include interface
 commands exec include configure terminal
 commands configure include interface FastEthernet0/0
!
parser view INTERFACE2
 secret CISCO
 commands interface include all ip
 commands configure include interface
 commands exec include configure terminal
 commands configure include interface FastEthernet0/1
!
parser view SUPER superview
 secret CISCO
 view DEBUG
 view INTERFACE1
 view INTERFACE2
```

Verification

Note

Role-based CLI enhances the local command authorization model. Instead of using hierarchical privilege levels, you may define user roles in Cisco IOS, called “views”, and assign command sets accessible to each view. You may further group views in hierarchies using the concept of a super-view that contains other views instead of a list of the allowed commands.

Each view is password-protected, and users are required to enter this password when switching to the view. Alternatively, a view may be associated with a user utilizing the local database or special external AAA attribute, specifying the role name. Note that enabling AAA with **`aaa new-model`** is mandatory for the role-based access-control to work.

One special “root” view always exists. By default, even when in privileged exec mode, you are not part of the root view. You need to explicitly switch to the root view to be able to create other views. This is the only difference from the regular enable privilege mode. You can switch to the root view using the command **`enable view`**.

You can define a new view using the command parser **`view <VIEW_NAME>`**. Under the view configuration mode, you define the password for the view using the command **`secret <password>`** and define commands accessible to the view using the syntax similar to command-authorization:

```
 commands parser-mode {include|exclude|include-exclusive}
[all] [interface interface-name] [command]
```

Here **`include-exclusive`** means the commands included in this view are not accessible to other views. By using the **`interface`** keyword you can limit the interfaces accessible to the role, provided that the user has access to the configure mode **`interface`** command. You can switch to the view using the command **`enable view <VIEW_NAME>`** in the router CLI.

If you need to define a superview, use the command **`parser view <NAME> superview`** and add views using the nested command **`view <SUBVIEW_NAME>.`** Note that all views, including superview, must have passwords defined in order to work.

To verify, try switching between the views in the router CLI. First try the view that has access only to one interface:

```
Rack1R4#enable view INTERFACE1
Password: CISCO

Rack1R4#
%PARSER-6-VIEW_SWITCH: successfully set to view 'INTERFACE1'.
Rack1R4#?
Exec commands:
  configure  Enter configuration mode
  enable     Turn on privileged commands
  exit       Exit from the EXEC
  show       Show running system information

Rack1R4#configure t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R4(config)#?
Configure commands:
  do         To run exec commands in config mode
  exit       Exit from configure mode
  interface  Select an interface to configure

Rack1R4(config)#interface fastEthernet 0/0
Rack1R4(config-if)#?
Interface configuration commands:
  exit  Exit from interface configuration mode
  ip    Interface Internet Protocol config commands

Rack1R4(config-if)#interface FastEthetnet 0/1
                                ^
% Invalid input detected at '^' marker.
```

Now try accessing the superview and check what commands are available:

```
Rack1R4#enable view SUPER
Password: CISCO

Rack1R4#?
Exec commands:
  configure  Enter configuration mode
  debug      Debugging functions (see also 'undebug')
  enable     Turn on privileged commands
  exit       Exit from the EXEC
  show       Show running system information
  undebug    Disable debugging functions (see also 'debug')
```

```
Rack1R4#
%PARSER-6-VIEW_SWITCH: successfully set to view 'SUPER'.
Rack1R4#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Rack1R4(config)#interface fastEthernet 0/0
Rack1R4(config-if)#?
Interface configuration commands:
  exit  Exit from interface configuration mode
  ip    Interface Internet Protocol config commands

Rack1R4(config-if)#interface fastEthernet 0/1
Rack1R4(config-if)#?
Interface configuration commands:
  exit  Exit from interface configuration mode
  ip    Interface Internet Protocol config commands

Rack1R4(config-if)#do sh run
Building configuration...

Current configuration : 171 bytes
!
!
!
!
!
!
interface FastEthernet0/0
 ip address 204.12.1.4 255.255.255.0
 ip rip advertise 10
!
interface FastEthernet0/1
 ip address 155.1.146.4 255.255.255.0
!
!
end

Rack1R4(config-if)#
```

11.31 IP Source Tracker

- You suspect that SW1 is under a distributed DoS attack from unidentified sources.
- Configure R6 to collect attack information for the VLAN 67 IP address of SW1.
- Export the statistics via syslog messages every 5 minutes.

Configuration

```
R6:
ip source-track syslog-interval 5
ip source-track 155.1.67.7
```

Verification

Note

IP source tracking offers a quick way to collect basic attack vector information for a given destination IP address. This is commonly needed in situations of distributed DoS attack where a single host is flooded by multiple sources. By enabling IP source tracking systematically on the path to the real source, you can backtrack every source of the attack to the network edge. There you can apply the actual filtering. The command to track a given destination IP address is:

`ip source-track <IP>`

The statistics collected include input interfaces and a packet count/packet rate. Based on this information you can track the next router closest to the attack source.

You may define the interval used by the feature to export the collected statistics as syslog entries using the command **`ip source-track syslog-interval`**.

To verify, generate some traffic from R4 towards the target IP address:

```
Rack1R4#ping 155.1.67.6 source fastEthernet 0/0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.67.6, timeout is 2 seconds:
Packet sent with a source address of 204.12.1.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms

Rack1R4#ping 155.1.67.7

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.67.7, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms

Rack1R4#ping 155.1.67.7 so fa 0/0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.67.7, timeout is 2 seconds:
Packet sent with a source address of 204.12.1.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms

Rack1R4#ping 155.1.67.7 so Loopback0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.67.7, timeout is 2 seconds:
Packet sent with a source address of 150.1.4.4
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms
Rack1R4#

Rack1R6#show ip source-track
Address         SrcIF          Bytes   Pkts     Bytes/s     Pkts/s
155.1.67.7      Fa0/0.146       1000     10          13          0
```

Note the aggregate statistics collected for the IP address, including the packet count, the total bytes switched, and the bytes per second rate.

11.32 Router IP Traffic Export

- Configure R1 to export the traffic sent and received on the Frame-Relay interface to the MAC address of R4.
- Configure a traffic filter that only allows TCP and ICMP packets to be monitored.

Configuration

```
R1:
ip access-list extended FILTER
 permit icmp any any
 permit tcp any any
!
ip traffic-export profile EXPORT
  interface FastEthernet0/0
  bidirectional
  incoming access-list FILTER
  outgoing access-list FILTER
  mac-address 0015.c634.6c61
 !
interface Serial 0/0
 ip traffic-export apply EXPORT
```

Verification

Note

Router IP traffic export (RITE) is a feature to monitor IP traffic received or sent on any WAN or LAN interface. This allows the monitoring of IP traffic on non-shared interfaces, such as Frame-Relay or ATM connections. Commonly this is needed for the purpose of IDS/IPS deployment or traffic monitoring.

One added benefit of using RITE is the ability to specify access-list filters for input or output traffic flows, allowing the router to select only the required traffic and minimize overhead.

RITE configuration consists of the following steps:

1) Defining an access-list to filter either inbound or outbound traffic or both.

2) Defining a RITE profile that specifies the exporting direction (inbound or bidirectional). By default the profile exports inbound traffic only.

3) Associate access-lists with inbound and outbound filters.

4) Define the export destination which is an output Ethernet interface and a MAC-address using the commands **`interface`** and **`mac-address`**.

RITE sends all IP packets unmodified and encapsulated into an Ethernet frame with the destination MAC address specified in the frame. Optionally, you may request the router to export every 1 of N packets in flows, in order to collect just statistical information and minimize the load on the router.

For this task, learn the MAC address of R4 using the command **`show ip arp 155.X.146.4`**.

You need a monitoring station with an Ethernet interface in promiscuous mode in order to capture the exported packets. From the router CLI, you can verify the configured traffic-export statistics.

Generate some ICMP traffic across R1 and check the statistics:

```
Rack1R6#ping 155.1.0.5 repeat 100

Type escape sequence to abort.
Sending 100, 100-byte ICMP Echos to 155.1.0.5, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max =
56/60/100 ms

Rack1R1#show ip traffic-export
Router IP Traffic Export Parameters
Monitored Interface             Serial0/0
        Export Interface                FastEthernet0/0
        Destination MAC address 0015.c634.6c61
        bi-directional traffic export is on
Output IP Traffic Export Information    Packets/Bytes Exported
100/10000
        Packets Dropped           0
        Sampling Rate             one-in-every 1 packets
        Access List        FILTER [named extended IP]
Input IP Traffic Export Information     Packets/Bytes Exported    0/0
        Packets Dropped           10
        Sampling Rate             one-in-every 1 packets
        Access List        FILTER [named extended IP]
        Profile EXPORT is Active
```

Note the increasing counter of exported packets. You may also check the increasing count of packets matching the filtering access-list:

```
Rack1R1#show ip access-lists FILTER
Extended IP access list FILTER
    10 permit icmp any any (200 matches)
    20 permit tcp any any
```

11.33 Controlling the ICMP Messages Rate

- In the future, you plan to apply an input traffic filter on R4's VLAN 146 interface.
- Configure this interface not to respond with an ICMP message when the filter drops a packet.
- The rate of these messages sent out of all other interfaces should not exceed 100 per second.
- In order to ensure that PMTUD works correctly, increase the rate of ICMP messages used by this feature to 1000 per second on R4.

Configuration

```
R4:
interface FastEthernet 0/1
 no ip unreachables
!
ip icmp rate-limit unreachable 10
ip icmp rate-limit unreachable DF 1
```

Verification

Note

Since routers operate at Layer 3, they can generate various ICMP messages when processing IP packets. Most commonly, routers respond with ICMP unreachable messages when dropping packets due to some reason. Most commonly the reasons are as follows:

1) Network/host/port unreachable - the router cannot route the packet to the destination.

2) Administratively Prohibited - the filter configuration on the router drops the packet.

3) Fragmentation required but DF-bit set - this message is critical in the PMTU discovery procedure.

There are two ways to control the generation of ICMP unreachable messages.

1) Disable generation of ICMP unreachables out of the specified interface. The interface-level command is **`no ip unreachables`**. You may want to enable this command in order to make it harder to map your network for an outside attacker by hiding the information of unreachable/filtered hosts or segments.

2) Control the rate of ICMP unreachable messages sent by the router. The global command is **`ip icmp rate-limit unreachable <once per this ms>`**. This command limits the total rate of all router-generated unreachable messages and prevents attacks designed to exhaust router resources.

3) Control the rate of "packet-too-big" messages (DF bit set but fragmentation required). Since this message type is crucial to PMTUD, you may want to set a separate rate-limit for it using the command **`ip icmp rate-limit unreachable DF <once per this ms>.`**

You can see if the unreachables are disabled on the interface using the following show command:

```
Rack1R4#show ip interface fastEthernet 0/1 | include unreach
  ICMP unreachables are never sent
```

Now try tracing the route to different R4 interfaces:

```
Rack1R1#traceroute 155.1.146.4 probe 5 timeout 1

Type escape sequence to abort.
Tracing the route to 155.1.146.4

  1  *  *  *  *  *
  2  *  *  *  *  *

Rack1R1#traceroute 155.1.0.4 probe 5 timeout 1

Type escape sequence to abort.
Tracing the route to 155.1.0.4

  1 155.1.0.5 32 msec 28 msec 28 msec 28 msec 32 msec
  2 155.1.0.4 69 msec 84 msec 96 msec 108 msec 116 msec
```

Traceroute sends UDP packets towards special UDP ports numbers. Since the router is the ultimate traceroute destination, it should respond with ICMP unreachables. In the first case, all probes timed out, since unreachables are disabled. In the second case, all probes are successful, since the router is configured to generate 100 messages per second.

11.34 Control Plane Policing

- Limit the aggregate rate of ARP traffic towards R1’s route processor to 100 packets per second.
- The routing control traffic marked with an IP Precedence value of 6 should be limited to 50 packets per second.
- Drop all Telnet connections sourced from R5’s Frame-Relay IP address.
- Limit the rate of outgoing ICMP messages to 10 per second.

Configuration

```
R1:
class-map ARP
 match protocol arp
!
ip access-list extended ICMP
 permit icmp any any
!
class-map ICMP
 match access-group name ICMP
!
ip access-list extended TELNET
 permit tcp host 155.1.0.5 any eq telnet
!
class-map TELNET
 match access-group name TELNET
!
class-map ROUTING
 match ip precedence 6
!
policy-map CPP_INPUT
 class ARP
  police rate 100 pps
 class TELNET
  drop
 class ROUTING
  police rate 50 pps
!
policy-map CPP_OUTPUT
 class ICMP
  police rate 10 pps
!
control-plane
 service-policy input CPP_INPUT
 service-policy output CPP_OUTPUT
```

Verification

Note

Control plane policing (CoPP) allows you to control the rate for traffic destined to or originated from the router process. This traffic includes all process-switched traffic, such as IP packets with options or packets logged by an access-list. Other types of traffic directed to the router process include routing updates, Telnet sessions, and any other traffic directed to the router itself.

The router may generate outgoing traffic, including ICMP responses, returning Telnet session traffic, outgoing IGP/BGP updates, outgoing Telnet sessions, etc.

In order to control this traffic, you need to define traffic classes using MQC syntax and apply a special control-plane policy-map. The only applicable policy-map actions are “drop” or “police”. For classification, you may use an IP access-list with DSCP/IP precedence matching. Additionally, you can match protocol ARP directly in the class-maps. Do not try to use NBAR to classify the control plane traffic, for it may have unpredictable results.

For the “police” action, CoPP supports a unique packet-per-second rate-limiting feature. Using the command **`police rate <N> pps`** you can specify the aggregate rate for a class in packets per second. You can optionally specify the burst size (amount of packets received instantly) if you need to fine-tune your configuration. Note that this feature only works with CoPP policy-maps.

To verify the CoPP feature, configure R4 to send ICMP packets to R1.

```
Rack1R4#ping 155.1.146.1 repeat 1000 timeout 1

Type escape sequence to abort.
Sending 1000, 100-byte ICMP Echos to 155.1.146.1, timeout is 1 seconds:
!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!
!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!
.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.!!.
Success rate is 66 percent (126/189), round-trip min/avg/max = 1/3/92
ms
```

In order to understand this behavior, look at the **`show policy-map control-plane`** command output at R1:

```
Rack1R1#show policy-map control-plane
 Control Plane

  Service-policy input: CPP_INPUT

    Class-map: ARP (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol arp
      police:
          rate 100 pps, burst 24 packets
        conformed 0 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 0 pps, exceed 0 pps

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group name TELNET
      drop

    Class-map: ROUTING (match-all)
      832 packets, 96612 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: ip precedence 6
      police:
          rate 50 pps, burst 12 packets
        conformed 832 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 1 pps, exceed 0 pps

    Class-map: class-default (match-any)
      238 packets, 28914 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

```
Service-policy output: CPP_OUTPUT

    Class-map: ICMP (match-all)
      207 packets, 23598 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group name ICMP
      police:
          rate 10 pps, burst 2 packets
        conformed 138 packets; actions:
          transmit
        exceeded 69 packets; actions:
          drop
        conformed 0 pps, exceed 0 pps

    Class-map: class-default (match-any)
      1013 packets, 119604 bytes
      5 minute offered rate 1000 bps, drop rate 0 bps
      Match: any
```

Note the default burst size of 10 pps – it is two packets. This is why R1 only accepts two packets in a row and drops the third packet. It takes one second for the token bucket to completely refill and then permit two more packets.

Now try ping and Telnet to R1 from R5's Frame-Relay IP address:

```
Rack1R5#telnet 155.1.0.1
Trying 155.1.0.1 ...
% Connection timed out; remote host not responding

Rack1R5#ping 155.1.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 72/199/392
ms
```

```
Rack1R1#show policy-map control-plane
 Control Plane

  Service-policy input: CPP_INPUT

    Class-map: ARP (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol arp
      police:
          rate 100 pps, burst 24 packets
        conformed 0 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 0 pps, exceed 0 pps

    Class-map: TELNET (match-all)
      4 packets, 192 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group name TELNET
      drop

    Class-map: ROUTING (match-all)
      1256 packets, 146110 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: ip precedence 6
      police:
          rate 50 pps, burst 12 packets
        conformed 1256 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 1 pps, exceed 0 pps

    Class-map: class-default (match-any)
      410 packets, 47646 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Note the four packets dropped under the class-map for the Telnet traffic from R5. The ping operation was 100% successful because the packet rate across the Frame-Relay cloud is less than 2 packets per second.

11.35 Control Plane Protection (CPPr)

- Enhance R3 protection by dropping packets going to all closed ports.
- Ensure this does not affect connections made on ports 3020 and 4040.
- To reduce the effect of broadcast storms, limit the rate of input non-IP packets to 100 per second
- Set the queue-limit for input HTTP packets to 100 packets and limit the packet rate to 10 per second.
- Ensure that all fragmented packets transiting the router are limited to 1 Mpps.

Configuration

Note

Control Plane Protection (CPPr) is Cisco IOS feature aimed at preventing infrastructure attacks, i.e. attacks targeting at the router itself. Control Plane implements routing and management protocols, such as OSPF, BGP, RIP, SNMP, SSH, Telnet and so on. The most typical attacks against control plane are of resource exhaustion type, i.e. they target at depleting router's resources and causing service denial. On most IOS platforms, control plane applications run on central Route Processor (or CPU) in parallel with asynchronous packet switching. Packet routing is commonly implemented using CEF switching path, during hardware interrupt processing task. All packets directed to the control plane (e.g. routing updates, keepalives, SSH/SNMP sessions) are handled using process-switching, which is most CPU intensive.

Since IOS 12.3T a feature known as Control Plane Policing allows for applying traffic policing for packets not handled during interrupt processing (fast/CEF paths). This feature permit basic protection against DoS attacks targeted at the router's CPU, such as fragmented ping flooding. A sample control plane policing configuration would look similar to the following example:

```
ip access-list extended BGP
 permit tcp any any eq bgp
!
ip access-list extended OSPF
 permit ospf any any
!
class-map BGP
 match access-group name BGP
!
class-map OSPF
 match access-group name OSPF
```

```
!
policy-map CONTROL_PLANE_POLICY
  class BGP
    police rate 2000 pps burst 100 packets
  class OSPF
    police rate 10 pps burst 3
  class class-default
    police rate 5000 pps burst 100 packets
!
control-plane
  service-policy input CONTROL_PLANE_POLICY
```

Notice that in the above configuration the police command specifies rate in packets per second and the burst size in packets. This type of the police command is only applicable to the control-plane policy. In addition to the input policy, you can configure output policing as well, and limit the rate of the packets produced by the router's control plane.

Using just the demonstrated aggregate form of policing along, you may already increase the level of your router's protections. However, CPPr extended CPP and made it even more granular. CPPr treat the Route Processor (RP) as a virtual interface attached to the router. All packets redirected to the RP for some reason are classified into three categories corresponding to three subinterfaces of the virtual interface:

`Control-plane host subinterface`. This interface receives all control-plane TCP/UDP traffic that is directly destined for one of the router interfaces. Examples of control-plane host traffic include management traffic or routing protocols. Notice that non TCP/UDP control traffic will end up on the CEF-exception sub-interface. Most control plane protection features operate on this subinterface and thus this sub-interface provides most features, such as policing, port filtering and per-protocol queue thresholds.

Port-filtering allows for automatically dropping of packets destined for the TCP/UDP ports not currently open in the router. The operating system automatically detects all open ports, and you can manually configure some exceptions. The net effect is reduced load on router's CPU during the times of flooding attacks.

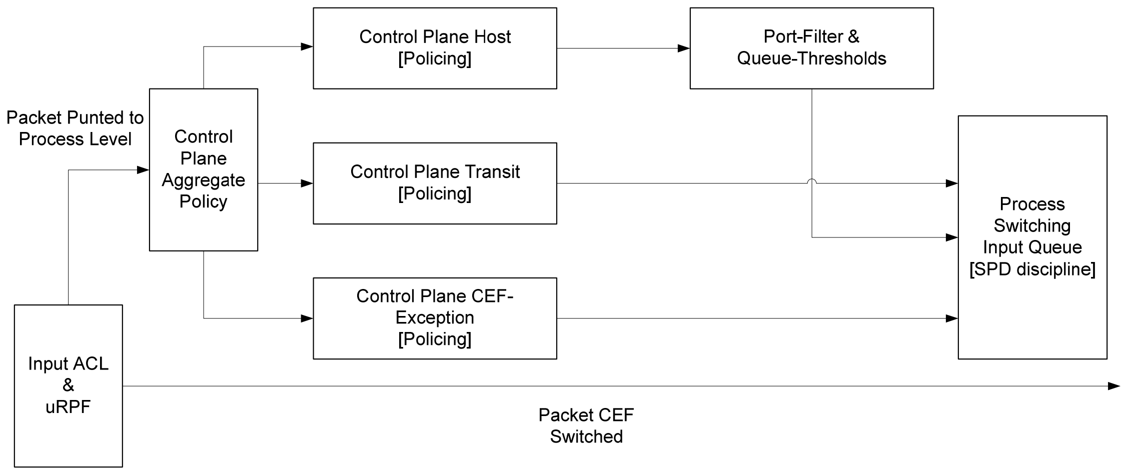
Per-protocol queue thresholds allow you setting selective queue limits for packets of different type, for example BGP, OSPF, FTP, DNS, HTTP, IGMP, SNMP etc.

`Control-plane transit subinterface`. This subinterface is intended to handle *transit* IP packets not handled via CEF mechanism and needed to be switching in the processor context. This might happen, for instance, when a packet mast be routed out of Ethernet interface and no ARP lookup has been made yet for the next-hop, making the CEF adjacency incomplete.

`Control-plane CEF exception subinterface`. This is where packets causing an exception in CEF switching path land. So far, those include non-IP router-destined traffic, such as CDP, L2 keepalive messages (e.g. Frame-Relay LMI keepalives) or ARP packets and IP packets with options or TTL <=1. Additionally, non-UDP local multicast traffic destined to the router (e.g. OSPF updates) fall into this category as well.

You may apply a separate rate-limiting policy to any of the sub-interface or have a single aggregate policy embracing all subinterfaces (classic control plane policing). It is possible to configure both the subinterface and aggregate policy, but this feature appears to work unstable in many 12.4T images. So far, it seems to be better configuring either aggregate or subinterface specific policies.

Before any packet reaches any specific control plane subinterface, it is first processed via a series of ingress features including input access-list, uRP checks, aggregate control-plane policy. After passing the subinterface-specific policy, the packet is then being enqueued onto the respective interface input queue and handled via SPD (selective packet discard) policy. Here is a visual outline of the process:


Control Plane Host [Policing]
Port-Filter & Queue-Thresholds
Packet Punted to Process Level
Control Plane Aggregate Policy
Control Plane Transit [Policing]
Process Switching Input Queue [SPD discipline]
Control Plane CEF-Exception [Policing]
Input ACL & uRPF
Packet CEF Switched

Let’s go and look into CPPr configuration steps. You start by defining class-maps to select control-plane traffic. Those are regular L3/L4 class-maps, matching access-lists/DSCP values etc. You cannot use NBAR for control-plane traffic classification. After this, you define a regular policy map and assign classes to it, applying the desired actions. The policing command is the same as described above, based on packet rate per second. However, after this the things start to differ. You may attach the new policy map to any of the control plane sub-interfaces, like this:

]

```
policy-map CPP_HOST_POLICY
  class BGP
    police rate 256000
!
control-plane host
  service-policy input CPP_HOST_POLICY
```

You may use the command **control-plane {host| cef-exception| transit}** to select any of the control-plane subinterfaces. Notice that you cannot configure egress policing for any of the host subinterfaces. Now, the most important host subinterface also supports the port-filtering feature. To configure this one, you first need to create a class-map defining ports to be filtered. The command is **class-map type port-filter [match-all | match-any] <NAME>.** For example

```
 class-map type port-filter match-any CLOSED_PORTS
  match closed-ports
  match port TCP 1024
  match port UDP 2048
```

The first **match** command auto-detects all open ports in the system and select all other ports as “closed”. To check the ports currently detected as “open” use the command **show control-plane host open ports**. As you can see, you may add your own ports using the match port statement. You can also use the **match not port** command to unblock a port that is not automatically detected by the system as open. For example, if you have a rotary-group 30 configured on a VTY line, you may want to make sure that TCP port 3030 is not blocked. You can use the following configuration:

```
class-map type port-filter match-all CLOSED_PORTS
  match closed-ports
  match not port 3020
```

To apply the port filtering settings, create a port-filtering policy-map and apply it to the host control-plane subinterface:

```
policy-map type port-filter FILTER_CLOSED_PORTS
  class CLOSED_PORTS
    drop
!
control-plane host
  service-policy type port-filter input FILTER_CLOSED_PORTS
```

This configuration will effectively drop all packets destined to closed ports before affecting router's CPU. The last feature to discuss is per-protocol input queue thresholds. As mentioned, those allow for differentiated per-protocol services on the control-plane host interface. In order to define a queue threshold, create a special queue-threshold class-map first. This class-map should match one or many of the supported protocols:

```
class-map type queue-threshold [match-all | match-any]
<CLASS>
 match protocol [bgp | dns | ftp | http | igmp | snmp | ssh
| syslog | telnet | tftp]
```

As you can see, the list is pretty extensive. You can use the special command **`match host-protocols`** to select ALL TCP/UDP based protocol running on the router and set a queue threshold for those. Packets are classified when they enter the router and are redirected to the host subinterface. Afte you have the special class-map defined you may configure a special queue-threshold policy-map and apply it to the host subinterface:

```
class-map type queue-threshold CMAP_BGP
  match protocol bgp
!
policy-map type queue-threshold BGP_THRESHOLD
  class CMAP_BGP
    queue-limit 50
!
control-plane host
 service-policy type queue-threshold input BGP_THRESHOLD
```

The queue limit is set in packets. This is the separate limit enforced for this particular protocol in the common input queue. There could be only one input queue-threshold policy-map, but you can configure every class with a separate limit.

R3:
```
!
! ACL to select fragmented IP packets
!
ip access-list extended FRAGMENTS
 permit ip any any fragment
!
class-map FRAGMENTS
 match access-group name FRAGMENTS

!
! HTTP traffic
!
ip access-list extended HTTP
 permit tcp any any eq www
!
class-map HTTP
 match access-group name HTTP

!
! Class-map to select closed ports. Notice that we
! need to unblock the RIP port explicitly.
!
class-map type port-filter match-all CLOSED_PORTS
 match closed-ports
 match not port TCP 3020
 match not port TCP 3040
 match not port UDP 520
!
policy-map type port-filter HOST_PORT_FILTER
 class CLOSED_PORTS
  drop

!
! To set the queue thresholds for HTTP
!
class-map type queue-threshold QUEUE_HTTP
 match protocol http
!
policy-map type queue-threshold HOST_QUEUE_THRESHOLD
  class QUEUE_HTTP
    queue-limit 100

!
! Control plane rate-limit policies
!
policy-map HOST_RATE_LIMIT
 class HTTP
  police rate 10 pps burst 2 packets

!
policy-map CEF_EXCEPTION_RATE_LIMIT
 class class-default
  police rate 100 pps burst 20 packets
```

```
!
policy-map TRANSIT_RATE_LIMIT
 class FRAGMENTS
  police rate 1000000 pps burst 200000 packets

!
! Applying the policies together
!
control-plane host
  service-policy input HOST_RATE_LIMIT
  service-policy type queue-threshold input HOST_QUEUE_THRESHOLD
  service-policy type port-filter input HOST_PORT_FILTER
!
control-plane transit
  service-policy input TRANSIT_RATE_LIMIT
!
control-plane cef-exception
 service-policy input CEF_EXCEPTION_RATE_LIMIT
```

Verification

Note

Configure an HTTP server in R3 and simulate an HTTP connection from R1 to R3.

```
Rack1R1#telnet 150.1.3.3 80
Trying 150.1.3.3, 80 ... Open
GET /

WWW-Authenticate: Basic realm="level_15_access"

401 Unauthorized

[Connection to 150.1.3.3 closed by foreign host]
Rack1R1#
```

Note

Check the policy-map statistics for the host subinterface.

```
Rack1R3#show policy-map control-plane host
 Control Plane Host

  Service-policy input: HOST_RATE_LIMIT

    Class-map: HTTP (match-all)
      12 packets, 720 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group name HTTP
      police:
          rate 10 pps, burst 2 packets
        conformed 8 packets; actions:
          transmit
        exceeded 4 packets; actions:
          drop
        conformed 0 pps, exceed 0 pps

    Class-map: class-default (match-any)
      5 packets, 338 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Note

Now check the port-filter policy map applied to the host subinterface. Configure R3 to listen on port 3020 by setting up a rotary group numer on any VTY line. Notice that the list of auto-detected open ports does not include the ports manually added to the policy-map.

```
Rack1R3#show policy-map type port-filter control-plane host
 Control Plane Host

  Service-policy port-filter input: HOST_PORT_FILTER

    Class-map: CLOSED_PORTS (match-all)
      4 packets, 543 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  closed-ports
      Match: not  port tcp 3020
      Match: not  port tcp 3040
      Match: not  port udp 520
      drop

    Class-map: class-default (match-any)
      12 packets, 816 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

R3:

```
line vty 0
 rotary 20
```

```
Rack1R3#show control-plane host open-ports
Active internet connections (servers and established)
Prot        Local Address      Foreign Address                  Service
State
 tcp                 *:23                   *:0                   Telnet
LISTEN
 tcp                 *:80                   *:0                HTTP CORE
LISTEN
 udp                 *:67                   *:0            DHCPD Receive
LISTEN
 udp                 *:68                   *:0             BootP client
LISTEN

Rack1R1#telnet 150.1.3.3 3020
Trying 150.1.3.3, 3020 ... Open


User Access Verification

Password:
```

Note

Check the queue thresholds for HTTP traffic class next. Notice that there are matches because we connected to the router using HTTP.

```
Rack1R3#show policy-map type queue-threshold control-plane host
      queue-limit 100
      queue-count 0     packets allowed/dropped 0/0
 Control Plane Host

  Service-policy queue-threshold input: HOST_QUEUE_THRESHOLD

    Class-map: QUEUE_HTTP (match-all)
      8 packets, 480 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  protocol http

    Class-map: class-default (match-any)
      18 packets, 1224 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Note

Next, check the CEF-exceptions subinterface statistics. There is a lot of packets matching this sub-interface as all non-IP keepalives, CDP packets, non-IP multicast hit this interface.

```
Rack1R3#show policy-map control-plane cef-exception
 Control Plane Cef-exception

  Service-policy input: CEF_EXCEPTION_RATE_LIMIT

    Class-map: class-default (match-any)
      537 packets, 38156 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          rate 100 pps, burst 20 packets
        conformed 537 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 1 pps, exceed 0 pps
```

Note

There is just one packet recorded for the transit control-plane subinterface. This packet was generated when we were pinging off R1 across R3 and the firewall attempted to resolve the IP to MAC mapping for R2.

```
Rack1R3#show policy-map control-plane transit
 Control Plane Transit

  Service-policy input: TRANSIT_RATE_LIMIT

    Class-map: FRAGMENTS (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group name FRAGMENTS
      police:
          rate 1000000 pps, burst 200000 packets
        conformed 0 packets; actions:
          transmit
        exceeded 0 packets; actions:
          drop
        conformed 0 pps, exceed 0 pps

    Class-map: class-default (match-any)
      1 packets, 114 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

11.36 IOS ACL Selective IP Option Drop

- Configure R3 to drop any packets destined to the router with IP source route option (either loose or strict).
- Apply this protection to the link connected to SW1.

Configuration

Note

IP options are special extension to the standard IP header (which is 20 bytes in length) that signal special processing requirements for the datagram. For example, IP options may instruct the router to record the route in the IP header or specify a source-route at the origin. Packets with IP options are process-switches by Cisco IOS routers and put significant load on the router CPU. This, it is important to filter the packets with IP options unless they are really needed. You may configure the router to silently discard all packets with IP options using the command **`ip options drop`**. You may use access-lists to drop IP options selectively, using the ACL line syntax similar to the following:

```
permit ip any any option <Option Name>
```

You may find the list of IP options using the command-line interface help features ("?" sign). The special **`any-option`** keyword selects any option. The access-list may be used in either a policy-map or as an access-group. Notice that only named extended access-lists are supported for this feature.

```
R3:
ip access-list extended DROP_IP_SOURCE_ROUTE
 deny ip any any option lsr
 deny ip any any option ssr
 permit ip any any
!
interface FastEthernet 0/0
 ip access-group DROP_IP_SOURCE_ROUTE in
```

Verification

Note

To verify IP option based filtering, generate ICMP packets with Loose source route option from SW1. Notice the response packets from R3 – the ICMP unreachable messages have reason “Received Packet has Options”.

```
Rack1SW1#ping
Protocol [ip]:
Target IP address: 155.1.37.3
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: Loose
Source route: 10.0.0.3
Loose, Strict, Record, Timestamp, Verbose[LV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.37.3, timeout is 2 seconds:
Packet has IP options:  Total option bytes= 7, padded length=8
 Loose source route: <*>
   (155.1.37.3)

Unreachable from 155.1.37.3.  Received packet has options
 Total option bytes= 7, padded length=8
 Loose source route: <*>
   (155.1.37.3)

<snip>
```

Note

Check the access-list statistics on R3 after this.

```
Rack1R3#show ip access-lists DROP_IP_SOURCE_ROUTE
Extended IP access list DROP_IP_SOURCE_ROUTE
    10 deny ip any any option lsr (5 matches)
    20 deny ip any any option ssr
    30 permit ip any any (164 matches)
```

11.37 BGP Generic TTL Security Mechanism

- Configure eBGP session between R5 and R6 and make both routers accept the peer if it's no more than one hop away.
- Use AS numbers 500 and 600 for R5 and R6 respectively.

Configuration

Note

General TTL Security Mechanism (GTSM) defined in RFC 3682 specifies a protection method against BGP session hijacking and resource exhaustion attacks. Generally, BGP process listens on the TCP port 139 and accepts all TCP SYN packets destined to this port, unless they are filtered by an ACL. It is possible to generate a barrage of spoofed packets imitating a valid BGP session and inject false information (if the session is unauthenticated) or generate a TCP SYN-flooding attack.

GTSM utilizes the simple fact that every router on the path to the BGP speaker decrements the TTL field in IP packets by one. Based on this, it is possible to identify potentially spoofed packets by looking at their TTL field – the packets send from "afar" will have the TTL field below some threshold. It is possible to define a "secure radius" in the number of hop counts to accept the incoming IP packets. For example, if all BGP peers are within 10 hops away from the local BGP speaker, then all incoming IP packets will have their TTL field set to at no less than 245. This is because all IP packets start with TTL=255 and the field is decremented by every hop on the path. Thus, by accepting the IP packets with TTL greater than or equal to 245 it is possible to minimize the risk of spoofed packets reaching the BGP process. Notice that the usefulness of GTSM feature decreases as the diameter of eBGP Multihop session grows.

In order to configure the TTL security checks for a BGP peer use the command **`neighbor <IP> ttl-security hops <hop-count>`**. This command applies to eBGP peering sessions only (either directly-connected or multihop) and specifies the number of hops the remote peer could be away from the local speaker. Keep in mind the internal BGP sessions are not protected, and therefore the internal network assumed to be "trusted". All packets incoming TCP packets targeted at BGP port with the IP TTL value below (255 - <hop-count>) are silently discarded by the router. In addition, the feature sets TTL value for outgoing TCP/IP packets to 255-<hop-count> to make sure the remote peer will accept the local packets. The GTSM feature is mutually exclusive with the **`ebgp-multihop`** BGP feature. This is because the eBGP session by default sets TTL=1 in the outgoing IP packets and with the **`multihop <n>`** session parameter, the TTL value is set to <n>, which is not compatible with GTSM

Therefore, make sure you configured GTSM feature on both sides of the peering link.

```
R5:
router bgp 100
 neighbor 155.1.6.6 remote-as 200
 neighbor 155.1.6.6 ttl-security hops 2

R6:
router bgp 200
 neighbor 150.1.5.5 remote-as 100
 neighbor 150.1.5.5 ttl-security hops 2
```

Verification

Note

Check the TTL settings for every peer. Notice the minimum incoming TTL of 253 in the peer properties.

```
Rack1R5#show ip bgp neighbors 150.1.6.6 | inc TTL
Connection is ECN Disabled, Mininum incoming TTL 253, Outgoing TTL 255

Rack1R6#show ip bgp neighbors 150.1.5.5 | include TTL
Connection is ECN Disabled, Mininum incoming TTL 253, Outgoing TTL 255
```

11.38 Flexible Packet Matching

- Configure R1 to filter ICMP Echo packets with the string “AAA” in the payload. Look no deeper than 256 bytes in the packet.
- Ensure the filtering applies only to ICMP/IP packets received in Ethernet frames.

Configuration

Note

Flexible Packet Matching is a new feature that allows for granular packet inspection in Cisco IOS routers. Using FPM you can match any string, byte or even bit at any position in the IP (or theoretically non-IP) packet. This may greatly aid in identifying and blocking network attacks using static patterns found in the attack traffic. This feature has some limitation though.

a) First, it is completely stateless, e.g. does not track the state/history of the packet flow. Thus, FPM cannot discover dynamic protocol ports such as use by H.323 or FTP nor cannot it detect patterns split across multiple packets. Essentially, you are allowed to apply inspection per-packet basis only.

b) Additionally, you cannot apply FPM to the control-plane traffic, as the feature is implemented purely in CEF switching layer. Fragmented traffic is not assembled for matching, and the only inspected packet is the initial fragment of the IP packet flow.

c) IP packets with IP options are not matched by FPM as well, because they are punted to the route processor.

d) Lastly, this feature inspects only unicast packets and does not apply to MPLS encapsulated packets.

Configuring an FPM filter consists of a few steps.

(1) Loading protocol headers.

(2) Defining a protocol stack.

(3) Defining a traffic filter.

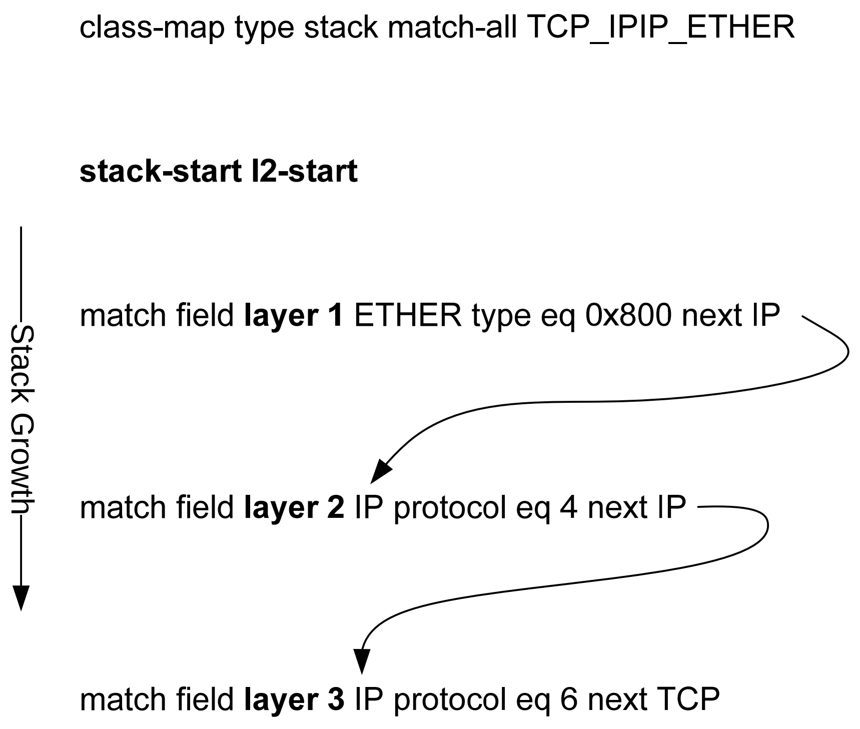
(4) Applying the policy & Verifying

Let’s look at every one of these steps in details.

`Load a PHDF (optional).` PHDF stands for Packet Header Definition file. Those files use XML syntax and define the structure of various packet headers, such as Ethernet, IP, TCP, UDP and so on. They are very helpful in making filtering more structured, as with the PHDFs loaded you may filter based on the header field names and their values, instead of matching fixed offsets in the unstructured packet body. You may load the files into the router's memory using the command **`load protocol <path>`**. The PDHF files could be manually created using the simple XML syntax or downloaded from CCO. Since IOS version 12.4(15)T, four basic PHDFs are included in the IOS code and located at the virtual path **`system:fpm/phdfs`**. You could load the files directly from there. Defining a custom PHDF requires understanding of protocol header formats and field values along with basic XML formatting, which is beyond the scope of this document.

`Define a protocol stack (optional)`. This step uses the PDHFs loaded previously and allows specifying the protocol headers found in the traffic you want to inspect. Using the protocol stack definition induces structure in the packets being inspected. This allows for filtering based on header field values and specifying offsets in the packet relative to the header fields. Additionally, you may define various protocol stacks (e.g. UDP in IP, UDP in GRE in IP) and reuse the same access control policy with various stacks.

You define the protocol stack using the command **`class-map type-stack match-all <NAME>`**. This class-map it is always of type match-all and consists of a series of the match entries. Every match entry should specify a protocol name defined in loaded PHDFs, a header field value and the next protocol name found in stack. Look at the sample below – it defines the series of headers: TCP in IPIP tunnel headed (protocol 4) encapsulated in IP and in the Ethernet.



```
class-map type stack match-all TCP_IPIP_ETHER

stack-start l2-start

match field layer 1 ETHER type eq 0x800 next IP

match field layer 2 IP protocol eq 4 next IP

match field layer 3 IP protocol eq 6 next TCP
```

The order of the match statements is important – it defines the actual header stacking along with the "layer x" keywords. The layers are counted bottom up starting at the stack bottom. The other important thing is that every next **`match`** statement should define the protocol specified in the **`next`** clause of the previous match statement. This is a basic consistency check illustrated using the arrows in the diagram.

A few words about the **`match`** operator. This command uses the syntax **`match field <PROTO> <header-field> <operator> <value>.`** It is important that the protocol and header fields are learned dynamically from PHDFS and this allows for high flexibility. The operator could be "eq", "neq", "range", "string", "regex", "lt" and "gt". The operator names are self-explanatory, but the "eq" operator also supports a special "mask" parameter. The mask is interpreted as a bitmap, where "1" means the corresponding bit in the value could be any of "0" or "1" and "0" means the respective bit is "locked".

By default, the protocol stack is matched stating AFTER the data-link level header. That is, the first header defined in the stack is match against the header going after the L2 header. In real life, this is commonly the IP header. However, sometime you may want to match the layer2 header fields as well, e.g. Ethernet addresses. To make the protocol stack match starting at L2 level, use the command **stack-start l2-start**.

At this point, having just stack class-maps you may already apply traffic filtering to the packets matching the configured stack. To accomplish this, create a policy-map of type “access-control” and assign a stack class to the policy-map. For example:

```
class-map type stack TCP_IN_IP_IN_ETHER
 statck-start l2-start
 match field ETHER type eq 0x800 next IP
 match field layer 2 IP protocol eq 0x6 next TCP
!
policy-map type access-control DROP_TCP_IN_IP_IN_ETHER
 class TCP_IN_IP_IN_ETHER
   drop
```

There is basically just one action available with the access-control policy-maps, and the action is drop. You may also use the send-response command under a class to make the router send an ICMP unreachable response. However, using just stack class-maps might be inflexible, as you cannot match the packets payload. You may however use this for filtering based on addresses, protocol flags and so on.

Define a traffic filter. Traffic filter is defined by means of special class-map of type “access-control” and configuring a respective policy-map of the same type. Using this type of class-maps you can match the protocol field values using the command **match field <PROTO>** if the respective protocol’s PHDF has been loaded. If you match the protocol fields, than the policy-map using the newly defined access-control class-map must be nested under the stack-type class-map defining this protocol. We’ll see an example later.

In addition to matching the protocol header fields, you can match the packet payload at a fixed offset against a fixed value, value range, string or regular expression. You may base the offset off the define protocol header field (e.g. +10 bytes from TCP flags) using the command **match start <PROTO> <FIELD-NAME> offset <OFFSET> size <SIZE>**. For example:

```
match start TCP checksum offset 100 size 1
match start IP version offset 0 size 1
```

Of course, this type of offset basing is only possible if the respective protocol definition has been loaded and the containing stack-type class map has this protocol defined.

Irrespective of the protocols loaded/defined, you may base the offset from the L2 or L3 packet starts (absolute offsets). For example if the packet is IP in Ethernet, than L2-start is the first bit of the Ethernet header and L3-start is the first bit of the IP header. The command syntax is as follows:

```
match start {l2-start|l3-start} offset <OFFSET-BYTES> size
<SIZE-BYTES> <operator> <value>
```

The command specifies the offset in bytes from the selected start. If the size if less than or equal to 4 bytes, you can use the “eq, neq, lt, gt” operators in addition to “regex” and “string”. This allows for per-bit matching using the “eq” and “mask” operators combination. For example:

```
match start l3-start offset 0 size 1 eq 0x2 mask 0x3
match start l2-start offset 36 size 5 string ABCDE
```

Create an access-control policy-map. There are two options here. You may create a simple access-control policy-map, which has just the basic access-control assigned, without nesting. For example:

```
class-map type access-control PASSWORD
 match start l3-start offset 0 size 100 regex
”.*[pP][aA][sS][wW].*”
!
policy-map type access-control DROP_PASSWORD
 class PASSWORD
  drop
```

When using this simple non-nested syntax, you may only use the absolute offsets with the command **`match {l2-start|l3-start}`** and cannot reference any protocol header fields.

You may use a more flexible approach, by nesting the filtering policy under a stack-type class-map configured in containing policy-map. This allows for using protocol headers in filtering policy or basing the offsets from the packet header fields. Notice that the nested filtering policy may only use the protocol headers defined in the containing stack class-map. Here is an example that looks for string “TEST” in TCP packets:

```
class-map type stack TCP_IN_IP
 match field IP protocol eq 0x6 next TCP
!
! Protocol fields matched in the filtering policy
! must be for the protocols defined in the stack
! class-map
!
class-map type access-control match-any FILTER_CLASS
 match start TCP payload offset 0 size 4 string TEST
!
policy-map type access-control FILTER_POLICY
 class FILTER_CLASS
  drop
!
policy-map type access-control STACK_POLICY
  class TCP_IN_IP
  service-policy FILTER_POLICY
```

`Apply the traffic filtering policy`. Finally, the access-control policy map should be applied to an interface either inbound or outbound using the interface-level command **`service-policy type access-control {input|output} <NAME>`**. This could be a simple non-nested policy using just stack-type class-maps or access-control class-maps or more advanced, stack and nested access-control policy-map.

R1:
```
!
! Load the protocol header definition
!
load protocol system:fpm/phdf/ip.phdf
load protocol system:fpm/phdf/icmp.phdf
load protocol system:fpm/phdf/tcp.phdf
load protocol system:fpm/phdf/udp.phdf
load protocol system:fpm/phdf/ether.phdf

!
! Define a protocol stack
!
class-map type stack ICMP_IN_IP_IN_ETHER
 stack-start l2-start
 match field ether type eq 0x800 next ip
 match field layer 2 ip protocol eq 1 next icmp

!
! Define access-control policy-map
!
class-map type access-control match-all ICMP_ECHO_STRING
 match field icmp type eq 8
 match start icmp payload offset 0 size 256 regex ".*AAAA.*"
!
policy-map type access ACCESS_CONTROL_POLICY
 class ICMP_ECHO_STRING
  log
!
policy-map type access-control STACK_POLICY
 class ICMP_IN_IP_IN_ETHER
   service-policy ACCESS_CONTROL_POLICY
!
interface FastEthernet 0/0
 service-policy type access-control input STACK_POLICY
```

Verification

Note

Generate ICMP packets with the payload “AAAA” from R4 to R1. To accomplish this, use any ASCII codes table to find that the ASCII code for “A” is 0x41. After this, issue the following command from R2:

```
Rack1R4#ping 150.1.1.1 source loopback 0 data 4141

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 150.1.4.4
Packet has data pattern 0x4141
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
```

Note

Now check the access-control policy-map statistics in R1. Notice that both the stack-map and the access-control map have been matched. Additionally, you should get the log message that appears after the debugging output:

```
Rack1R1#show policy-map type access-control interface fastEthernet 0/0
 FastEthernet0/0

  Service-policy access-control input: STACK_POLICY

    Class-map: ICMP_IN_IP_IN_ETHER (match-all)
      5 packets, 570 bytes
      5 minute offered rate 0 bps
      Match: field ETHER type eq 0x800 next IP
      Match: field IP protocol eq 1 next ICMP

      Service-policy access-control : ACCESS_CONTROL_POLICY

        Class-map: ICMP_ECHO_STRING (match-all)
          5 packets, 570 bytes
          5 minute offered rate 0 bps
          Match: field ICMP type eq 8
          Match: start ICMP payload-start offset 0 size 256 regex
".*AAAA.*"
      log

        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any

    Class-map: class-default (match-any)
      1 packets, 86 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

%SEC-6-IPACCESSLOGDP: list ICMP_ECHO_STRING permitted icmp 150.1.4.4
(FastEthernet0/0 ) -> 150.1.1.1 (8/0), 5 packets
```

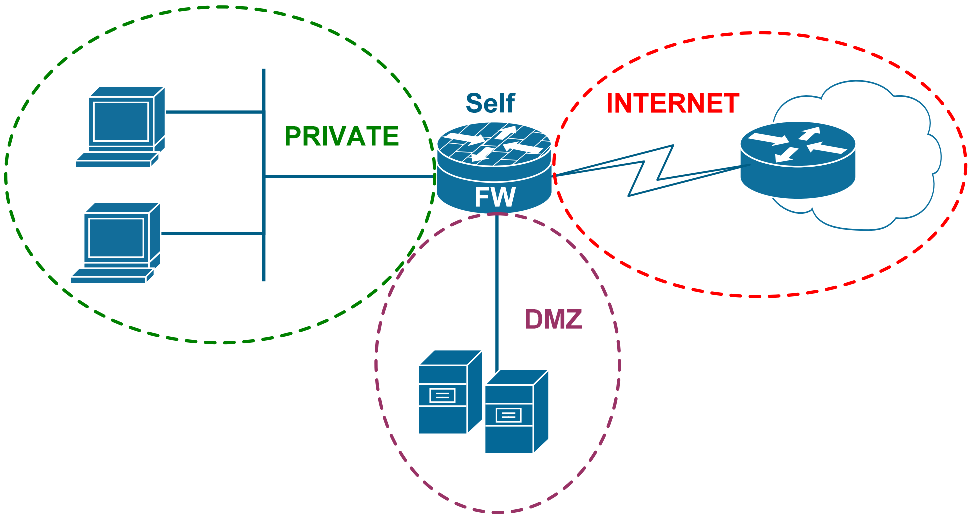
11.39 Zone Based Firewall

- Configure three security zones in R3: INSIDE, OUTSIDE and DMZ for the interfaces facing R1, R2 and SW1 respectively.
- Only allow R3 to be accessed using SSH and HTTPs from zones OUTSIDE and DMZ.
- Allow the INSIDE users to use the following protocols when accessing OUTSIDE: HTTP, FTP, ICMP, DNS, SSH, Telnet, AOL Instant Messenger.
- Account for the AOL Messenger using the non-standard port 80 and users connecting to HTTP proxies on ports 3128 and 8080.
- Allow the OUTSIDE users to access the servers in DMZ using the HTTP, FTP, DNS and TACACS+ protocols.
- The INSIDE users should have the same set of protocols allowed to zone DMZ with the addition of SSH and HTTPS.
- Preserve IP routing with your configuration and allow the users logged into the router to ping/telnet to OUTSIDE and DMZ zones.
- Limit the inside users to subnet 150.X.1.0/24 and the DMZ server to the subnet 155.X.37.0/24.

Configuration

Note

Zone based Firewall of ZFW is a new approach to configuring access control in the IOS firewall. Prior to this feature, traffic filtering was accomplished using access-lists and stateful traffic inspection rules (CBAC). Both the access-lists and inspection rules apply directly to the physical interfaces, which may poorly reflect organization’s security policy, as the policy deals with more high-level objects than interfaces. A new core concept of ZFW is zone, which groups different interfaces sharing the same security attributes, the same level of trust. For example, you may have security zones that reflect your enterprise security levels partitioning. On the figure below, you can see three security zones assigned to three router interfaces:



Permissions for traffic forwarding is made between the zones, not physical interfaces. By default, traffic is permitted between the interfaces within the same security zone, and blocked between different zones. Traffic between the interface configured in a security zone and interface not in any zone is blocked. In addition, you cannot apply classic firewall rules (e.g. access-lists or inspection rules) to the interface configured in a security zone.

There is one default zone in every router, known as **`self`**, which encompasses the router's own IP addresses. Traffic to and from this zone to any other zone is *permitted* by default, to allow for control plane and management plane traffic. However, you may apply an explicit policy between the **`self`** zone and any other configured zone to control the router-originated traffic. If you apply a policy from any configured zone to the **`self`** zone, the traffic from **`self`** zone to the other zone will be permitted, but the returning traffic from the configured zone may be blocked. To resolve this, you may need to assign a policy from zone self to the configured zone and inspect the router-generated traffic.

In order to permit certain traffic flows between two different security zones, you need to define a zone pair and apply an inspection policy to this pair. Zone pairs are ordered, i.e. {A, B} is not the same as {B, A}. The first zone in the pair is source zone and the other one is destination zone. When you assign the inspection policy to a zone pair, it applies to the traffic flowing from source to destination zone. The inspection policy is configured as a set of traffic-classes along with associated actions, such as inspect, pass or drop. Traffic classes are generally defined using access-lists and/or protocol matching.

Zone based firewall implements the same set of features as the classic IOS firewall. It allows defining advanced inspection options, such as deep protocol inspection and parameter tuning. To accomplish all listed features, ZFW uses a new configuration framework called Cisco Policy Language (CPL). The syntax for the new configuration language was heavily borrowed from the well-known MQC (Modular QoS CLI) and resembles it in many ways, by using policy-maps, class-maps and actions association. Here is a brief overview of ZFW configuration steps:

1) Define zones. After initial planning, you define security regions and come with the zone names. The command to accomplish this is **`zone security <NAME>`** and every zone may have a meaningful description assigned to it.

2) Define zone-pairs. Zone pairs should be defined prior to policy configuration. As mentioned before, zone pairs are ordered sets and specify the direction of traffic flow. The command to define a zone pair is: **`zone-pair security <NAME> source <A> destination <B>`**, e.g. **`zone-pair security SELF_TO_OUTSIDE source self destination OUTSIDE`**.

3) Define class-maps that describe traffic that must have policy applied as it crosses a zone-pair. There is a new special type of class-maps used at this step – **`class-map type inspect`**, which specifies inspection scope.

4) Define policy-maps to apply action to your class-maps' traffic. A new special type of policy-map called **`policy-map type inspect`** has been introduced by CPL to specify inter-zone policies. You assign the previously defined class-maps to the policy-map and specify associated actions.

5) Define other advanced policy attributes. Advanced options include inspection parameter tuning, TCP/UDP session limiting and application inspection rules. We'll discuss that in separate scenario.

6) Apply policy-maps to zone-pairs. The command to apply the security policy to a zone pair is the same as to define the pair. For example:

```
zone-pair security INSIDE_TO_SELF source INSIDE destination
self
  service-policy type inspect PMAP_INSIDE_TO_SELF
```

Notice that policy applies just in one direction, and in some cases you may need to apply a policy in the opposite direction as well.

7) Assign interfaces to zones. This is last step, accomplished using the interface command **`zone-member security <ZONE>`**.

Defining Traffic Classes.

Generic inspection type class-maps could be of type **`match-all`** or **`match-any`**, just like the regular MQC class-maps. In the first case, all match conditions must be met in order for the class-map to match the traffic. You can match an access-list using the command **`match access-group [number] [name <NAME>]`**, which is the only way to select traffic sources or destination.

In addition to matching access-lists, you can match protocols supported by the inspection engine. The list of the protocols is the same as supported by CBAC. Unlike the classic class-map, when you configure the **`match protocol`** command, you do not engage NBAR engine – rather, the protocol will be selected for inspection once the containing policy-map will be assigned to a zone pair. It is common to combine access-list matching with protocol matching, for example:

```
class-map match-all CMAP_HTTP_INSIDE_USERS
 match access-group name ACL_INSIDE_USERS
 match protocol http
```

would match HTTP traffic for inside users. You can use the command **`show ip port-map`** to list all supported protocols available for matching.

You may apply class-map nesting using the command **`match class-map`**. This may be vefy helpful in situations where you need to apply complex AND/OR logic. For example, if you want to select a set of protocols used (e.g. HTTP, DNS, FTP, ICMP) used by a group of users. You may not combine multiple protocols match and ACL matching within a single class-map, but you may use class-map nesting as follows:

```
class-map match-any CMAP_INSIDE_PROTOCOLS
 match protocol http
 match protocol ftp
 match protocol dns
 match protocol icmp
```

```
!
ip access-list extended ACL_INSIDE_USERS
 perm ip 192.168.0.0 0.0.0.255 any
!
class-map match all CMAP_INSIDE_USERS
 match class-map CMAP_INSIDE_PROTOCOLS
 match access-group name INSIDE_USERS
```

Applying Policy Actions.

There are three general policy actions applicable under inspect-type policy-maps: “inspect”, “drop” and “pass”. The first action is similar to the CBAC inspection rule. It allows for stateful inspection of traffic flowing from source to destination zone, and automatically permits returning traffic flows even for complex protocols, such as H.323. The “drop” action simple discards matching traffic, and the “pass” action permits packet flow without stateful inspection, similar to “permit” action in access-lists. Here is an example:

```
policy-map PMAP_OUTSIDE_TO_DMZ
 class-map CMAP_LEGIT_TRAFFIC
    inspect
```

Notice that if you are using the “inspect” action, then the class-map must hav at least once “match protocol” statement, to specify the protocols for inspection. Otherwise, all protocols will be inspected. If you are using the “pass” action, make sure the returning traffic is also permitted, as the pinholes will not be opened dynamically.

Every inspection policy-map has an implicit class-default configured with action “drop”. This simulates the implicit “deny any” action found in traditional access-lists.

Port-Mapping.

As mentioned previously, ZFW supports the same protocol set as CBAC. You may see the port-numbers used by various applications by issuing the command **show ip port-map**. If you need to define a custom protocol port, you may use the command **ip port-map <protocol>** to assign new port numbers to the protocol. If you need to re-use the well-know port for a different application, e.g. assign telnet application to port 80, you need to use an access-list using the command **ip port-map telnet port <PORT> list <ACL>.** These are the same requirement as with the classic CBAC firewall. For example:

```
access-list 99 permit 192.168.1.0 0.0.0.255
!
ip port-map telnet port tcp 80 list 99
```

Self-Zone Considerations.

Self-zone policy has limited functionality as compared to the policies available for transit-traffic zone-pairs. First, for the stateful inspection, router-generated traffic is limited to TCP, UDP, ICMP, and complex-protocol inspection for H.323. Application Inspection (HTTP, FTP, Telnet) is not available for self-zone policies, i.e. you cannot apply any advanced protocol filtering for traffic destined to/from the firewall. Additionally, session and rate limiting cannot be configured on self-zone policies.

You may want to configure stateful inspection for protocol terminated/originated at the router if you apply a policy between the zone self and any other configured zone. The policy-map by default drops all non-matched traffic, and this may cause issues. For example, consider that you configured to pass only ICMP and SSH traffic to the firewall using the following configuration

```
ip access-list extended ACL_OUTSIDE_TO_SELF
 permit tcp any any eq 22
 permit icmp any any
!
class-map type inspect match-any CMAP_OUTSIDE_TO_SELF
 match access-group name ACL_OUTSIDE_TO_SELF
!
policy-map type inspect PMAP_OUTSIDE_TO_SELF
 class CMAP_OUTSIDE_TO_SELF
   pass
!
zone-pair security ZP_OUTSIDE_TO_SELF source OUTSIDE
destination self
 service-policy type inspect PMAP_OUTSIDE_TO_SELF
```

This will effectively block any ICMP/TCP/UDP sessions originated off the router, by the virtue of implicit class-default found in the new policy-map. In order to resolve this issue, you may configure the following inspection (remember that only UDP/TCP/ICMP/H.323 are supported, everything else should be permitted explicitly).

```
class-map type inspect match-any  CMAP_ROUTER_PROTOCOLS
 match protocol icmp
 match protocol udp
 match protocol tcp
!
policy-map type inspect PMAP_SELF_TO_OUTSIDE
 class CMAP_ROUTER_PROTOCOLS
  inspect
!
zone-pair security ZP_SELF_TO_OUTSIDE source self
destination OUTSIDE
 service-policy type inspect PMAP_SELF_TO_OUTSIDE
```

If you are using OSPF for routing and have a policy to control traffic from outside to self, you may need to configure something like the following:

```
ip access-list extended OSPF
 permit ospf any any
!
class-map type inspect CMAP_OSPF
 match access-group name OSPF
!
policy-map type inspect PMAP_OUTSIE_TO_SELF
 class CMAP_OSPF
   pass
```

If you want to have at least some inspection for protocols not supported with the zone self, you may still use TCP/UDP inspection. For example, if you want added security with HTTPS and SSH you may use the following configuration to apply basic traffic inspection:

```
ip access-list extended ACL_SSH_HTTPS
 permit tcp any any eq 22
 permit tcp any any eq 443
!
class-map type inspect match-all CMAP_SSH_HTTPS
 match access-group name ACL_SSH_HTTPS
 match protocol tcp
!
policy-map type inspect PMAP_OUTSIDE_TO_SELF
 class CMAP_SSH_HTTPS
   inspect
```

Note the above configuration is only really needed if you have a policy-map applied to the {self, OUTSIDE} zone pair and thus may block traffic returning from the router.

Finally, here is a short lists of the ZFW configuration steps in the recommended order.

1) Define access-lists for traffic scopes
2) Define class-maps
3) Define policy-maps
4) Configure zones and zone-pairs, apply the policies
5) Assign zones to the interfaces

Try using meaningful names for all object, and use prefixes ACL_, CMAP_, PMAP_ to allow you easily distinguishing object's purpose.

R3:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Access-Lists
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
! Traffic sourced FROM INSIDE IPs
!
ip access-list standard ACL_INSIDE_USERS
 permit 150.1.1.0 0.0.0.255

!
! Traffic TO DMZ IPs
!
ip access-list extended ACL_DMZ_HOSTS
 permit ip any 155.1.37.0 0.0.0.255

!
! Telnet sessions (for zone self inspection)
!
ip access-list extended ACL_TELNET
 permit tcp any any eq 23

!
! SSH and HTTPs connections
!
ip access-list extended ACL_SSH_HTTPS
 permit tcp any any eq 22
 permit tcp any any eq 443

!
! RIP updates
!
ip access-list extended ACL_RIP
 permit udp any any eq 520

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Class-maps of type inspect
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
! RIP updates
!
class-map type inspect match-all CMAP_RIP
 match access-group name ACL_RIP

!
! HTTPs and SSH traffic
!
class-map type inspect match-all CMAP_HTTPS_SSH
 match access-group name ACL_SSH_HTTPS
 match protocol tcp

!
! Telnet Traffic
!
class-map type inspect match-all CMAP_TELNET
```

```
 match access-group name ACL_TELNET
 match protocol tcp

!
! ICMP Traffic
!
class-map type inspect match-all CMAP_ICMP
 match protocol icmp

!
! Protocols allowed from INSIDE to OUTSIDE
!
class-map type inspect match-any CMAP_INSIDE_TO_OUTSIDE_PROTOCOLS
 match protocol http
 match protocol ftp
 match protocol icmp
 match protocol dns
 match protocol ssh
 match protocol telnet
 match protocol aol

!
! Protocols allowed from OUTSIDE to DMZ
!
class-map type inspect match-any CMAP_OUTSIDE_TO_DMZ_PROTOCOLS
 match protocol http
 match protocol ftp
 match protocol dns
 match protocol tacacs

!
! Protocols allowed from INSIDE to DMZ
!
class-map type inspect match-any CMAP_INSIDE_TO_DMZ_PROTOCOLS
 match protocol http
 match protocol ftp
 match protocol dns
 match protocol tacacs
 match protocol ssh
 match protocol https

!
! Traffic from INSIDE to OUTSIDE
!
class-map type inspect match-all CMAP_INSIDE_TO_OUTSIDE_ACCESS
 match access-group name ACL_INSIDE_USERS
 match class-map CMAP_INSIDE_TO_OUTSIDE_PROTOCOLS

!
! Traffic from OUTSIDE to DMZ
!
class-map type inspect match-all CMAP_OUTSIDE_TO_DMZ_ACCESS
 match class-map CMAP_OUTSIDE_TO_DMZ_PROTOCOLS
 match access-group name ACL_DMZ_HOSTS
!
! Traffic from INSIDE to DMZ (note the two ACLs used)
!
```

```
class-map type inspect match-all CMAP_INSIDE_TO_DMZ_ACCESS
 match class-map CMAP_INSIDE_TO_DMZ_PROTOCOLS
 match access-group name ACL_DMZ_HOSTS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Policy-Maps for Configured Zones
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
! Policy-maps for INSIDE, OUTSIDE, DMZ
!
policy-map type inspect PMAP_INSIDE_TO_OUTSIDE
 class CMAP_INSIDE_TO_OUTSIDE_ACCESS
  inspect
!
policy-map type inspect PMAP_INSIDE_TO_DMZ
 class CMAP_INSIDE_TO_DMZ_ACCESS
  inspect
!
policy-map type inspect PMAP_OUTSIDE_TO_DMZ
 class CMAP_OUTSIDE_TO_DMZ_ACCESS
  inspect

!
! Policy-maps dealing with the zone self.
!
policy-map type inspect PMAP_OUTSIDE_TO_SELF
 class CMAP_HTTPS_SSH
  inspect
 class CMAP_RIP
  pass
!
policy-map type inspect PMAP_DMZ_TO_SELF
 class CMAP_HTTPS_SSH
  inspect
!
policy-map type inspect PMAP_SELF_TO_ANY
 class CMAP_TELNET
  no pass
  inspect
 class CMAP_ICMP
  no pass
  inspect
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Zones and Zone-Pairs
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

zone security INSIDE
zone security OUTSIDE
zone security DMZ
!
zone-pair security ZP_INSIDE_TO_OUTSIDE source INSIDE destination
OUTSIDE
 service-policy type inspect PMAP_INSIDE_TO_OUTSIDE
!
zone-pair security ZP_INSIDE_TO_DMZ source INSIDE destination DMZ
 service-policy type inspect  PMAP_INSIDE_TO_DMZ
!
zone-pair security ZP_OUTSIDE_TO_DMZ source OUTSIDE destination DMZ
 service-policy type inspect  PMAP_OUTSIDE_TO_DMZ
!
zone-pair security ZP_OUTSIDE_TO_SELF source OUTSIDE destination self
 service-policy type inspect  PMAP_OUTSIDE_TO_SELF
!
zone-pair security ZP_DMZ_TO_SELF source DMZ destination self
 service-policy type inspect PMAP_DMZ_TO_SELF
!
zone-pair security ZP_SELF_TO_OUTSIDE source self destination OUTSIDE
 service-policy type inspect PMAP_SELF_TO_ANY
!
zone-pair security ZP_SELF_TO_DMZ source self destination DMZ
 service-policy type inspect PMAP_SELF_TO_ANY
!
! Apply zones to interfaces
!
interface Serial 1/2
 zone-member security INSIDE
!
interface Serial 1/3
 zone-member security OUTSIDE
!
interface FastEthernet 0/0
 zone-member security DMZ
```

Verification

Note

To test the Zone Based Firewall configuration, generate traffic flows between different zones. Start with the INSIDE to OUTSIDE direction. Use the **`show policy-map`** command on R3 to display statistics for the respective policy and look for packet matches. There are other protocols inspected by the policy, but we don't test them in this verification.

```
Rack1R1#ping 150.1.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms
Rack1R1#

Rack1R3#show policy-map type inspect zone-pair ZP_INSIDE_TO_OUTSIDE
 Zone-pair: ZP_INSIDE_TO_OUTSIDE

  Service-policy inspect : PMAP_INSIDE_TO_OUTSIDE

    Class-map: CMAP_INSIDE_TO_OUTSIDE_ACCESS (match-all)
      Match: access-group name ACL_INSIDE_USERS
      Match: class-map match-any CMAP_INSIDE_TO_OUTSIDE_PROTOCOLS
        Match: protocol http
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ftp
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol icmp
          1 packets, 80 bytes
          30 second rate 0 bps
        Match: protocol dns
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ssh
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol telnet
          1 packets, 24 bytes
          30 second rate 0 bps
        Match: protocol aol
          0 packets, 0 bytes
          30 second rate 0 bps
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [0:41]
        icmp packets: [0:10]
        Session creations since subsystem startup or last reset 2
        Current session counts (estab/half-open/terminating) [1:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:00:08
        Last statistic reset never
        Last session creation rate 2
        Maxever session creation rate 2
        Last half-open session total 0

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        6 packets, 368 bytes
```

Note

While telnet session is still connected, check the sessions inspected by the ZFW. Notice the session corresponding to your telnet connection.

```
Rack1R3#show policy-map type inspect zone-pair ZP_INSIDE_TO_OUTSIDE
sessions
 Zone-pair: ZP_INSIDE_TO_OUTSIDE

  Service-policy inspect : PMAP_INSIDE_TO_OUTSIDE

    Class-map: CMAP_INSIDE_TO_OUTSIDE_ACCESS (match-all)
      Match: access-group name ACL_INSIDE_USERS
      Match: class-map match-any CMAP_INSIDE_TO_OUTSIDE_PROTOCOLS
        Match: protocol http
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ftp
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol icmp
          1 packets, 80 bytes
          30 second rate 0 bps
        Match: protocol dns
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ssh
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol telnet
          2 packets, 48 bytes
          30 second rate 0 bps
        Match: protocol aol
          0 packets, 0 bytes
          30 second rate 0 bps
      Inspect
        Established Sessions
         Session 844E2620 (155.1.13.1:17513)=>(150.1.2.2:23) telnet
SIS_OPEN
```

```
          Created 00:01:02, Last heard 00:01:00
          Bytes sent (initiator:responder) [37:79]

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        6 packets, 368 bytes
```

Note

Now perform the similar tests for connections from OUTSIDE and INSIDE to the DMZ zone. Generate some traffic first...

```
Rack1R1#telnet 155.1.37.7 80
Trying 155.1.37.7, 80 ... Open
GET /

HTTP/1.1 200 OK
<snip>

Rack1R2#telnet 155.1.37.7 80
Trying 155.1.37.7, 80 ... Open
GET /
HTTP/1.1 200 OK
<snip>
```

Note

Check the policy-map statistics after this:

```
Rack1R3#show policy-map type inspect zone-pair ZP_INSIDE_TO_DMZ
 Zone-pair: ZP_INSIDE_TO_DMZ

  Service-policy inspect : PMAP_INSIDE_TO_DMZ

    Class-map: CMAP_INSIDE_TO_DMZ_ACCESS (match-all)
      Match: access-group name ACL_DMZ_HOSTS
      Match: class-map match-any CMAP_INSIDE_TO_DMZ_PROTOCOLS
        Match: protocol http
          1 packets, 24 bytes
          30 second rate 0 bps
        Match: protocol ftp
          3 packets, 72 bytes
          30 second rate 0 bps
        Match: protocol dns
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol tacacs
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ssh
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol https
          0 packets, 0 bytes
          30 second rate 0 bps
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [6:38]
        ftp packets: [0:1]

        Session creations since subsystem startup or last reset 2
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:01:35
        Last statistic reset never
        Last session creation rate 0
        Maxever session creation rate 1
        Last half-open session total 0

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        10 packets, 688 bytes
```

```
Rack1R3#show policy-map type inspect zone-pair ZP_OUTSIDE_TO_DMZ
 Zone-pair: ZP_OUTSIDE_TO_DMZ

  Service-policy inspect : PMAP_OUTSIDE_TO_DMZ

    Class-map: CMAP_OUTSIDE_TO_DMZ_ACCESS (match-all)
      Match: class-map match-any CMAP_OUTSIDE_TO_DMZ_PROTOCOLS
        Match: protocol http
          1 packets, 24 bytes
          30 second rate 0 bps
        Match: protocol ftp
          3 packets, 72 bytes
          30 second rate 0 bps
        Match: protocol dns
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol tacacs
          0 packets, 0 bytes
          30 second rate 0 bps
      Match: access-group name ACL_DMZ_HOSTS
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [3:32]
        ftp packets: [0:1]

        Session creations since subsystem startup or last reset 2
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:01:07
        Last statistic reset never
        Last session creation rate 0
        Maxever session creation rate 2
        Last half-open session total 0

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        0 packets, 0 bytes
```

Note

Finally, we check the inspection of the router-generated traffic. Per the configuration, users should be able to ping and telnet off the router. We test this assumption by generating traffic to the OUTSIDE zone.

```
Rack1R3#ping 150.1.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/8 ms

Rack1R3#telnet 150.1.2.2
Trying 150.1.2.2 ... Open


User Access Verification

Password:
Rack1R2>exit

[Connection to 150.1.2.2 closed by foreign host]
```

Note

Check the matches for the policy-map applied to traffic from zone self to the zone OUTSIDE. Notice that there was one session created per every class.

```
Rack1R3#show policy-map type inspect zone-pair ZP_SELF_TO_OUTSIDE
 Zone-pair: ZP_SELF_TO_OUTSIDE

  Service-policy inspect : PMAP_SELF_TO_ANY

    Class-map: CMAP_TELNET (match-all)
      Match: access-group name ACL_TELNET
      Match: protocol tcp
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [40:0]

        Session creations since subsystem startup or last reset 1
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:00:07
        Last statistic reset never
        Last session creation rate 1
        Maxever session creation rate 1
        Last half-open session total 0
    Class-map: CMAP_ICMP (match-all)
      Match: protocol icmp
      Inspect
        Packet inspection statistics [process switch:fast switch]
        icmp packets: [20:0]

        Session creations since subsystem startup or last reset 2
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:0]
        Last session created 00:00:10
        Last statistic reset never
        Last session creation rate 1
        Maxever session creation rate 1
        Last half-open session total 0

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        0 packets, 0 bytes
```

Note

Now, make sure that the users on the OUTSIDE zone can access the router using SSH but not telnet. Additionally, make sure that the OUTSIDE users cannot ping the firewall.

```
Rack1R2#ping 155.1.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 155.1.23.3, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)

Rack1R2#telnet 155.1.23.3
Trying 155.1.23.3 ...
% Connection timed out; remote host not responding

Rack1R2#telnet 155.1.23.3 22
Trying 155.1.23.3, 22 ... Open
SSH-1.99-Cisco-1.25

Rack1R3#show policy-map type inspect zone-pair ZP_OUTSIDE_TO_SELF
 Zone-pair: ZP_OUTSIDE_TO_SELF

  Service-policy inspect : PMAP_OUTSIDE_TO_SELF

    Class-map: CMAP_HTTPS_SSH (match-all)
      Match: access-group name ACL_SSH_HTTPS
      Match: protocol tcp
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [11:0]

        Session creations since subsystem startup or last reset 1
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:00:10
        Last statistic reset never
        Last session creation rate 1
        Maxever session creation rate 1
        Last half-open session total 0

    Class-map: CMAP_RIP (match-all)
      Match: access-group name ACL_RIP
      Pass
        0 packets, 0 bytes

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        20 packets, 1144 bytes
```

11.40 ZFW Rate Limiting

- In order to prevent DoS attacks, limit the OUTSIDE to DMZ traffic flow to 512Kbps.
- Enable half-open session limiting, so that no more than 2000 half-open sessions are allowed and no more than 100 new half-open sessions generated per minute are permitted.
- Stop dropping the half-open session states once the absolute amount of half-open connections falls below 1000 and the rate falls below 10.

Configuration

Note

Zone based firewall supports two types of rate-limiting:

1) Limiting aggregate packet rate for the flows between security zones.
2) Limiting the maximum number and/or rate of the half-open connections for TCP/UDP sessions.

The first feature resembles QoS traffic policing, but applies to traffic matching the inspect class maps and essentially limits the aggregated flows between zones, and not out/in of a single interface. Thus if you have multiple interfaces in a single zone, the configuration will apply aggregate limiting across all interfaces. The command to apply policing is **`police rate <RATE Bps> burst <Burst in bytes>`** configured under the respective class of a policy-map. Any traffic bursts exceeding the configured rate are dropped by the firewall, i.e. you cannot apply advanced actions such as remarking. The smaller is the burst, the less traffic could be sent instantly after an idle period. By setting larger burst value, you ensure smoother traffic flows but risk getting sudden spikes of heavy traffic bursts. There is no optimal value for the burst value – they are usually picked up based on experimental data. Here is an example of ZFW policing configuration:

```
policy-map type inspect PMAP_OUTSIDE_TO_INSIDE
 class ICMP
   inspect
   police 256000 burst 8000
```

Notice that you need the inspect action configured prior to enabling the policing. One limitation of the traffic-rate policing is that it is not supported with the zone self – i.e. for any pair that includes this zone.

The other rate-limiting feature applies to the TCP/UDP based connections. It helps preventing resource exhaustion DoS attacks, such as SYN flooding and behaves similarly to TCP Intercept feature. However, the only mode available is “watch” mode where firewall passively observes connections and tears them apart when thresholds are crossed. The UDP sessions are emulated as bidirectional UDP packets exchange that time out after some configurable timer value.

In order to apply connection-based limits to a particular class applied under a policy map you need to define an inspect parameter-map first. This object defines various connection attributes including TCP-intercept like parameters and various timeouts. The values defined in the parameter-map are similar to the CBAC parameter values familiar to you from the classic firewall configuration. Here is a full-list of the inspect parameter-map configurable options.

```
parameter-map type inspect parameter-map-name
 alert {on|off}
 audit-trail {on|off}
 dns-timeout <seconds>
 icmp idle-timeout <seconds>
 max-incomplete {low <num-of-conn> | high <num-of-conn>}
 one-minute {low <num-of-conn> | high <num-of-conn>}
 tcp max-incomplete host <threshold> [block-time <minutes>]
 sessions maximum <sessions>
 tcp finwait-time <seconds>
 tcp idle-time <seconds>
 tcp synwait-time <seconds>
 udp idle-time <seconds>
```

The **`alert`** and **`audit-trail`** options define whether to produce an alert for particular event (e.g. security violation, improper TCP stream structure, etc) and generate connection statistics information, e.g. number of bytes send/received. The various timeout values (e.g. **`dns-timeout, tcp finwait-time, tcp idle-time`**) specify the amount of time to expire a particular type of connection.

The setting for TCP-intercept feature are:

`max-incomplete low/high` – specify the low (stop dropping half-open sessions) and high (start dropping half-open sessions) absolute thresholds for the half-open TCP/UDP connections. A half-open UDP connection is the connection that sent but not received any packet.
`one-minute low/high` – specify the per-minute thresholds for the number of session. This effectively controls the rate of the half-open connections.

`tcp max-incomplete host` – sets the number of half-open sessions allowed per host and the time to block this host after the limit has been exceeded.

In order to apply the parameter-map to a zone pair, use the option to the **`inspect`** command configured under a particular class. The traffic flows matching this class will inherit the parameters specified by the parameter map. Other classes will use the default parameter-map that could be viewed using the command **`show parameter-map type inspect default`**. Here is an example that limits the number of session to one hundred for the internal users:

```
parameter-map type inspect PMAP_PARAMS
  sessions maximum 100
!
policy-map type inspect PMAP_INSIDE_TO_OUTSIDE
  cass INSIDE_TRAFFIC
    inspect PMAP_PARAMS
```

R3:

```
parameter-map type inspect PMAP_PARAMS
 max-incomplete low 1000
 max-incomplete high 2000
 one-minute low 10
 one-minute high 100
 tcp max-incomplete host 200 block-time 1
 sessions maximum 5000
 dns-timeout 15

!
! The burst rate does not matter much, unless you have to
! tune the application performance in real-life scenarios.
!
policy-map type inspect PMAP_OUTSIDE_TO_DMZ
 class CMAP_OUTSIDE_TO_DMZ_ACCESS
  police rate 512000 burst 32000
  inspect PMAP_PARAMS
```

Verification

Note

First, check the default values configured in the system. Next, review the configured map and compare it with the default settings. Notice that the default values for the session interception feature are unlimited, meaning no session control is used.

```
Rack1R3#show parameter-map type inspect default
 parameter-map type inspect default values
  audit-trail on
  alert off
  max-incomplete low  unlimited
  max-incomplete high unlimited
  one-minute low  unlimited
  one-minute high unlimited
  udp idle-time 10
  icmp idle-time 5
  dns-timeout 0
  tcp idle-time 3600
  tcp finwait-time 5
  tcp synwait-time 30
  tcp max-incomplete host unlimited block-time 0
  sessions maximum 0

Rack1R3#show parameter-map type inspect
 parameter-map type inspect PMAP_PARAMS
  audit-trail on
  alert off
  max-incomplete low  1000
  max-incomplete high 2000
  one-minute low  10
  one-minute high 100
  udp idle-time 10
  icmp idle-time 5
  dns-timeout 15
  tcp idle-time 3600
  tcp finwait-time 5
  tcp synwait-time 30
  tcp max-incomplete host 200 block-time 1
  sessions maximum 5000

Rack1R3#show policy-map type inspect PMAP_OUTSIDE_TO_DMZ
  Policy Map type inspect PMAP_OUTSIDE_TO_DMZ
    Class CMAP_OUTSIDE_TO_DMZ_ACCESS
      Inspect PMAP_PARAMS
      Police rate 512000 burst 32000
    Class class-default
```

Note

Now add ICMP to the list of protocols inspected from OUTSIDE to DMZ and generate a flood of ICMP packets from OUTSIDE to the AAA server.

```
Rack1R3(config)#class-map type inspect CMAP_OUTSIDE_TO_DMZ_PROTOCOLS
Rack1R3(config-cmap)#match protocol icmp

Rack1R2#ping 155.1.37.7 size 1000 timeout 0 repeat 1000

Type escape sequence to abort.
Sending 1000, 1000-byte ICMP Echos to 155.1.37.7, timeout is 0 seconds:
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
....................
Success rate is 0 percent (0/1000)
Rack1R2#
```

Note

Now check the statistics for the policy applied to the respective zone pair. Notice that the policing statistics appears on top of the show command output. Approximately 10% of the ICMP packets exceeded the configured settings.

```
Rack1R3#show policy-map type inspect zone-pair ZP_OUTSIDE_TO_DMZ
 Zone-pair: ZP_OUTSIDE_TO_DMZ
      Police
      rate 512000 bps,32000 limit
      conformed 118 packets, 120124 bytes; actions: transmit
      exceeded 983 packets, 1000694 bytes; actions: drop
      conformed 0 bps, exceed 0 bps

  Service-policy inspect : PMAP_OUTSIDE_TO_DMZ

    Class-map: CMAP_OUTSIDE_TO_DMZ_ACCESS (match-all)
      Match: class-map match-any CMAP_OUTSIDE_TO_DMZ_PROTOCOLS
        Match: protocol http
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol ftp
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol dns
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol tacacs
          0 packets, 0 bytes
          30 second rate 0 bps
        Match: protocol icmp
          1 packets, 980 bytes
          30 second rate 0 bps
      Match: access-group name ACL_DMZ_HOSTS
      Inspect
        Packet inspection statistics [process switch:fast switch]
        tcp packets: [3:32]
        icmp packets: [0:118]
        ftp packets: [0:1]

        Session creations since subsystem startup or last reset 3
        Current session counts (estab/half-open/terminating) [1:0:0]
        Maxever session counts (estab/half-open/terminating) [1:1:1]
        Last session created 00:00:05
        Last statistic reset never
        Last session creation rate 1
        Maxever session creation rate 2
        Last half-open session total 0

    Class-map: class-default (match-any)
      Match: any
      Drop (default action)
        2000 packets, 1960000 bytes
```

11.41 ZFW Application Inspection

- Configure the firewall so that the internal users are not allowed to use insecure POP3/IMAP4 logins to the outside servers.
- Block image downloads from digg.com for internal users and log any attempts to use AOL messenger for non text-chat service (e.g. file transfer).
- Image files have filename extensions .jpg, .png, .gif
- Log all violations.

Configuration

✐ **Note**

Application inspection and control (AIC) applies at Layer 7 of the OSI model, allowing for policy configuration based on application protocol parameters. Applications are the most vulnerable part of your network, and thus enforcing strict per-application policy is important to increase the overall security. For example, you may want your firewall to block certain SMTP commands that open vulnerabilities in the servers. Alternatively, you may want to make sure that HTTP connections are not used to tunnel instant-messenger services or used to connect to other non-HTTP services. This is only possible with application-level inspection. Of course, there are numerous limitations for IOS AIC, if you compare this feature with ASA's MPF capabilities.

Cisco IOS Software ZFW offers application inspection and control on these application services:

```
HTTP
SMTP/ESMTP
POP3/IMAP
P2P Application Traffic (eDonkey, Kazaa2, GNUtella etc)
IM Applications
Sun RPC (MSN/AOL/Y!M)
```

AIC uses special type of class-maps and policy-maps to configure application protocol inspection. Those are *application specific* inspection class-maps and policy-maps for example **`class-map type inspect http, policy-map type inspect http`** and so on, for every application protocol supported. The special class-map defines protocol parameters and should be used inside the special policy map as usual. The special policy-map should be later applied as a *nested (child)* service-policy under the generic inspection policy-map. For example:

```
class-map type inspect http match-any CMAP_HTTP_VIOLATION
  match req-resp protocol-violation
!
policy-map type inspect http PMAP_HTTP_VIOLATION_RESET
  class type inspect http CMAP_HTTP_VIOLATION
   reset
!
class-map type inspect match-any CMAP_HTTP_TRAFFIC
 match protocol http
!
policy-map type inspect PMAP_INSIDE_OUTSIDE
 class type inspect CMAP_HTTP_TRAFFIC
  inspect
  service-policy http PMAP_HTTP_VIOLATION_RESET
```

It is important that the parent L3/L4 class-map (CMAP_HTTP_TRAFFIC in our case) define only the protocols supported by AIC. For example, if the parent class-map defines FTP protocol (**`match protocol ftp`**), you cannot apply the nested service-policy for any AIC supported protocol, as FTP is not supported by AIC.

Application inspection and control (AIC) has different set of features for every of the above-mentioned protocols.

1) HTTP inspection offers granular filtering on several types of application activity, including different methods (GET/POST), HTTP header contents and URIs. Additionally HTTP inspection allows you limiting document transfer size, web address lengths, and the flow of browser requests/server replies to enforce compliance with application-behavior standards. Finally, you may limit types of content (e.g. block java apps, images, videos or compressed content) that are transferred over HTTP connections.

2) SMTP inspection can limit content length and enforce protocol compliance. It checks for command syntax and permits only the legal SMTP/ESMTP commands.

3) POP3 and IMAP inspection can help ensure that users are using secure authentication mechanisms to prevent compromise of user credentials. This is illustrated in this particular scenario.

One special type of objects that is often used with the application inspection are regular-expressions parameter maps. It allows you defining sets of regular expressions to be later used with application inspection, for example to match URL strings or command parameters, such as FTP GET filename or SMTP recipient. The syntax to define a regular-expression parameter map is:

```
parameter-map type regex REGEX_CISCO
  pattern  [cC][iI][sS][cC][oO]
!
class-map type inspect http PMAP_CISCO
 match request header host regex REGEX_CISCO
```

When matching HTTP header values , don't forget the leading ".*" as Cisco IOS compiles the resulting regex with the header name prepended, e.g. "[hH][oO][sS][tT]:<pattern>" when you match the "host" header field against "<pattern>". Additionally, be careful when using the "$" anchor, as every HTTP line ends with \CR\LF and you would need to match those as well. Try not to use regex anchors at all, when matching various protocol header fields.

You may find the detailed information on the application inspection parameters in Cisco's documentation. From the lab perspective, make sure you have a good understanding of application filtering and basic application-specific feature, and refer to the documentation for additional information.

R3:

```
!
! Define regular expression first
!
parameter-map type regex REGEX_IMAGES
 pattern .*\.([jJ][pP][gG]|[pP][nN][gG]|[gG][iI][fF])
!
parameter-map type regex REGEX_DIGG
 pattern .*[dD][iI][gG][gG]\.[cC][oO][mM]

!
! The class that matches image URLs on Digg
!
class-map type inspect http match-all CMAP_DIGG_IMAGES
 match request header host regex REGEX_DIGG
 match request uri regex REGEX_IMAGES
 no match request arg regex REGEX_IMAGES

!
! Insecure POP3/IMAP4 logins
!
class-map type inspect pop3 CMAP_POP3_INSECURE
 match login clear-text
!
class-map type inspect imap CMAP_IMAP_INSECURE
 match login clear-text
```

```
!
! AOL Chat service
!
class-map type inspect aol CMAP_AOL_NON_CHAT
 match no service text-chat

!
! Policy-map type inspect IM could use MSN, Y!M and AOL
! type class-maps. They are all unified under a single policy
! of type IM
!
policy-map type inspect im PMAP_AOL_POLICY
 class CMAP_AOL_NON_CHAT
  log

policy-map type inspect http PMAP_HTTP_POLICY
 class CMAP_DIGG_IMAGES
  reset
  log
!
policy-map type inspect pop3 PMAP_POP3_POLICY
 class CMAP_POP3_INSECURE
  reset
  log
!
policy-map type inspect imap PMAP_IMAP_POLICY
 class CMAP_IMAP_INSECURE
  reset
  log

!
! We need to de-configure grouped protocol matching
! and create separate class-maps for every protocol
! that is to be inspected in-depth. This is to avoid
! possible collisions with protocols not supported by AIC
!
class-map type inspect CMAP_INSIDE_TO_OUTSIDE_PROTOCOLS
 no match protocol http
 no match protocol aol
 no match protocol pop3
 no match protocol imap

!
! Protocol-specific class-maps
!
class-map type inspect match-all CMAP_INSIDE_TO_OUTSIDE_HTTP
 match protocol http
 match access-group name ACL_INSIDE_USERS
!
class-map type inspect match-all CMAP_INSIDE_TO_OUTSIDE_AOL
 match protocol aol
 match access-group name ACL_INSIDE_USERS
!
class-map type inspect match-all CMAP_INSIDE_TO_OUTSIDE_POP3
 match protocol pop3
 match access-group name ACL_INSIDE_USERS
```

```
!
class-map type inspect match-all CMAP_INSIDE_TO_OUTSIDE_IMAP
 match protocol imap
 match access-group name ACL_INSIDE_USERS

!
! Modify the zone-pair policy-maps
!
policy-map type inspect PMAP_INSIDE_TO_OUTSIDE
 class CMAP_INSIDE_TO_OUTSIDE_HTTP
  inspect
  service-policy http PMAP_HTTP_POLICY
class CMAP_INSIDE_TO_OUTSIDE_AOL
  inspect
  service-policy im PMAP_AOL_POLICY
class CMAP_INSIDE_TO_OUTSIDE_POP3
  inspect
  service-policy pop3 PMAP_POP3_POLICY
class CMAP_INSIDE_TO_OUTSIDE_IMAP
  inspect
  service-policy imap PMAP_IMAP_POLICY
```

Verification

Note

We are going to test some of the application firewall capabilities. First, we configure R2 as HTTP server. Next, we emulate an HTTP GET request from R1 requesting the file “image.jpg” from the virtual host “digg.com”. Notice the message logged in R3:

```
Rack1R1#telnet 150.1.2.2 80
Trying 150.1.2.2, 80 ... Open
GET /image.jpg HTTP/1.1
Host: digg.com

[Connection to 150.1.2.2 closed by foreign host]

%APPFW-4-HTTP_URI_REGEX_MATCHED: URI regex
(.*\.([jJ][pP][gG]|[pP][nN][gG]|[gG][iI][fF])) matched - resetting
session 155.1.13.1:60627 150.1.2.2:80 on zone-pair ZP_INSIDE_TO_OUTSIDE
class CMAP_INSIDE_TO_OUTSIDE_HTTP appl-class CMAP_DIGG_IMAGES

%APPFW-4-HTTP_HDR_FIELD_REGEX_MATCHED: Header field
(^[Hh][Oo][Ss][Tt]:.*[dD][iI][gG][gG]\.[cC][oO][mM]) matched -
resetting session 155.1.13.1:60627 150.1.2.2:80 on zone-pair
ZP_INSIDE_TO_OUTSIDE class CMAP_INSIDE_TO_OUTSIDE_HTTP appl-class
CMAP_DIGG_IMAGES
```

Note

Now let’s try checking POP3 inspection. Since we don’t have a POP3 server running in R2, let’s use a trick. Enable TCP “echo” service in R2 and configure R3 to inspect POP3 on the TCP/7 port for the IP address 155.X.23.2. After this, connect from R1 to R2 on port 7 and issue POP3 “USER” command. Notice the debugging output in the firewall, showing that inspection encountered invalid command.

R3:

```
access-list 99 permit 155.1.23.2
!
ip port-map pop3 port tcp 7 list 99

Rack1R1#telnet 155.1.23.2 7
Trying 155.1.23.2, 7 ... Open
UUSSEERR  ppeetrtr

[Connection to 155.1.23.2 closed by foreign host]

%FW-5-POP3_INVALID_COMMAND: (target:class)-
(ZP_INSIDE_TO_OUTSIDE:CMAP_INSIDE_TO_OUTSIDE_POP3):Invalid POP3 command
from initiator (155.1.13.1:18292): Invalid verb
```

11.42- Classic IOS Transparent Firewall

- Configure R6 as a transparent firewall deployed on the subnet 10.0.0.0/24.
- Change the IP addresses for VLAN67 and VLAN146 interface of SW1, R1 and R4 to reflect this.
- VLAN67 users are allowed to originate HTTP, FTP, ICMP, DNS and SSH traffic across the firewall to VLAN146.
- VLAN146 users are only allowed initiating FTP (passive), HTTP and DNS connections to the serves on VLAN 67
- Do not permit any IPv6 traffic across the firewall.

Configuration

Note

IOS Transparent firewall was introduced in IOS version 12.3(7)T. It allows for traffic filtering and stateful inspection when the router acts as Layer 2 bridge. The benefit of using this operational mode is the ease of deployment – the end users are not required to change any networking settings – the firewall acts as the bump on a wire.

In transparent firewall mode, the router does not attempt any routing and simple learns Ethernet MAC addresses on all interfaces, performing Ethernet frame switching based on destination MAC addresses. In order to configure the IOS firewall as a bridge, you should first define a type of bridging configured in the firewall. This could be either CRB (Concurrent Routing and Bridging) or IRB (Integrated Routing and Bridging). Both use the concept of a bridge group, which is essentially a virtual bridge inside the router, with its own MAC address table.

1) CRB mode is enabled using the command **`bridge crb`**. Allows the router to bridge frames between interfaces configured as members of a single bridge group. Interfaces not in any bridge group with IP addresses assigned may accept and forward IP packets. Essentially, the router acts as a bridge for one set of interfaces and L3 router for the rest of the interfaces.

2) IRB mode is enabled using the command **`bridge irb`**. allows configuring a special BVI (Bridge Virtual Interface) for every bridge group. This interface represents the router as L3 device within the respective bridge group. All devices in the group aware of this BVI's MAC address may send IP packets to the router and communicate with other L2 domains. This mode allows the router to bridge between the interfaces and route packets for the members of the bridge-group that are aware of this service. You may compare BVI with an SVI (Switch Virtual Interface, e.g. VLAN 31) in a Layer 3 switch. In addition to the **`bridge irb`**

command you need to add the command **`bridge <number> route ip`** where <number> is the bridge-group number. Otherwise, the BVI interface will not route the IP packets.

In order to configure an interface/subinterface as member of a bridge group, use the command **`bridge-group <number>`** in interface configuration mode. By default, the new group starts with Spanning-Tree protocol disabled. You may need the command **`bridge <n> protocol IEEE`** to enable the IEEE-compatible spanning tree for the bridge-group. Notice that for 802.1Q subinterfaces the firewall will run the PVST+ variant of STP.

You may apply the classic firewall inspect rules and access-lists to the interface configured with the bridge-group numbers after this. For example:

```
bridge irb
bridge-group 1 protocol ieee
bridge 1 route ip
!
ip inspect name CBAC tcp
ip inspect name CBAC udp
ip inspect name CBAC icmp
!
ip acess-list extended OUTSIDE_IN
 deny ip any any log
!
interface FastEthernet 0/0
 bridge-group 1
 ip access-group OUTSIDE_IN in
!
interface FastEthernet 0/1
 bridge-group 1
 ip inspect CBAC in
```

The inspection rules apply to any support IP-based protocol. Only TCP/UDP/ICMP flows are supported by L2 transparent firewall. All non-IP packets are permitted by default across the firewall, including ARP, STP, IPv6 and so on. In order to block a non-IP protocol, you need to apply a protocol-type access-list to the bridged interface. Those lists have their numbers in range 201-299 and permit traffic based on Ethertype of SNAP PID value. For example, the following list will permit IP and ARP but block IPX:

```
access-list 201 permit 0x800
access-list 201 permit 0x806
access-list 201 deny 0x8137
access-list 201 permit 0x0 0xFFFF
```

The Ethertype for now becoming popular IPv6 is 0x86DD, and you can use this value to filter this non-IP protocol. In order to associate a protocol-type list with an interface, use the interface-level command **`bridge-group <n> input-type-list <ACL#>`**.

Transparent firewall does not inspect or process multicast Ethernet frames – they are permitted regardless of any input acces-lists. However, broadcast frames (e.g. NetBIOS over TCP/UDP or DHCP) are subject to transparent ACL checks. Transparent firewall provides special handling for DHCP packets as those are important for host auto-configuration. The command to permit DHCP packets across the firewall regardless of the IP access-lists configuration is **`ip inspect l2-transparent dhcp-passthrough`**.

Lastly, a few words on the IP address assigned to the BVI interface. It must be on the same subnet as the protected segments. You may treat is as equivalent to the ASA firewall's management IP. The traffic originated from the router across the bridged interface is not subject to the checks by the ACL applied on the bridged interfaces. Instead, to control the management access, you may need a separate access-list attached to the BVI.

The solution for this task configures inspection in two directions: inside to outside,dmz and outside to DMZ. Notice that the access-list applied to the outside interface needs to permit the RIP protocol updates and the access to the DMZ subnet. The access-list applied ingress to the DMZ interface prevents any unauthorized traffic from returning. The configuration does not look elegant, and this is a direct result of the inspection rules being applied per-interface, not per security-zone.

SW1:
```
interface Vlan 67
 ip address 10.0.0.7 255.255.255.0
```

R1:
```
interface FastEthernet 0/0
 ip address 10.0.0.1 255.255.255.0
```

R4:
```
interface FastEthernet 0/1
 ip address 10.0.0.4 255.255.255.0
!
! Enable SSH for testing
!
ip domain-name cisco.com
crypto key generate rsa general modulus 768
```

R6:
```
bridge irb
bridge 1 protocol ieee
bridge 1 route ip
!
interface BVI1
 ip address 10.0.0.6 255.255.255.0
!
ip inspect name INSIDE_PROTOCOLS http
ip inspect name INSIDE_PROTOCOLS ftp
ip inspect name INSIDE_PROTOCOLS dns
ip inspect name INSIDE_PROTOCOLS ssh
ip inspect name INSIDE_PROTOCOLS icmp
!
ip inspect name OUTSIDE_PROTOCOLS ftp
ip inspect name OUTSIDE_PROTOCOLS http
ip inspect name OUTSIDE_PROTOCOLS dns


!
! Outside ACL, need to permit access to INSIDE
!
no ip access-list extended OUTSIDE_IN
ip access-list extended OUTSIDE_IN
 permit tcp any any eq 80
 permit udp any any eq 43
 permit ftp any any eq 21
 deny ip any any log


!
! Block IPv6
!
access-list 201 deny 0x86dd
access-list 201 permit 0x0 0xFFFF

!
! Inside Interface
!
interface FastEthernet 0/0.67
 encapsulation dot1q 67
 no ip address
 bridge-group 1
 ip inspect INSIDE_PROTOCOLS in
 bridge-group 1 input-type-list 201

!
! Outside Interface
!
interface FastEthernet 0/1.146
 encapsulation dot1q 146
 no ip address
 bridge-group 1
 ip access-group OUTSIDE_IN in
 bridge-group 1 input-type-list 201
```

Verification

Note

Start verifications by checking the bridge-group status. Make sure you see MAC addresses learned at all interfaces configured for bridging and check the TX counters. It may take some time to learn MAC addresses, as this process requires traffic to be generated by devices.

```
Rack1R6#show bridge

Total of 300 station blocks, 297 free
Codes: P - permanent, S - self

Bridge Group 1:

    Address       Action   Interface       Age   RX count   TX count
0013.c451.f240   forward   Fa0/0.146         0        123         23
0013.c440.3980   forward   Fa0/0.67          0        297         23
```

Note

Check CBAC configuration next. We have two rules configured, one to inspect the internal sessions, and another to protect the DMZ servers. Notice that the DMZ rules do not allow ICMP traffic, but the inside rules do. Thus, the outside users are not allowed to ping the servers, while inside users can do this.

```
Rack1R6#show ip inspect config
Session audit trail is disabled
Session alert is enabled
one-minute (sampling period) thresholds are [unlimited : unlimited]
connections
max-incomplete sessions thresholds are [unlimited : unlimited]
max-incomplete tcp connections per host is unlimited. Block-time 0
minute.
tcp synwait-time is 30 sec -- tcp finwait-time is 5 sec
tcp idle-time is 3600 sec -- udp idle-time is 30 sec
tcp reassembly queue length 16; timeout 5 sec; memory-limit 1024 kilo
bytes
dns-timeout is 5 sec
Inspection Rule Configuration
 Inspection name INSIDE_PROTOCOLS
    http alert is on audit-trail is off timeout 3600
    ftp alert is on audit-trail is off timeout 3600
    dns alert is on audit-trail is off timeout 30
    ssh alert is on audit-trail is off timeout 30
    icmp alert is on audit-trail is off timeout 10
```

Note

Now test the firewall rules. Telnet from R4 to SW1 and confirm that the session is not established. At the same time, ensure that you can connect to R4 on SSH port and ping it R4 from SW1.

```
Rack1R4#telnet 10.0.0.7
Trying 10.0.0.7 ...
% Connection timed out; remote host not responding

R3:
%SEC-6-IPACCESSLOGP: list OUTSIDE_IN denied tcp 10.0.0.4(23) ->
10.0.0.7(19184), 1 packet

Rack1SW1#telnet 10.0.0.4 22
Trying 10.0.0.2, 22 ... Open

Rack1R6#show ip inspect sessions
Established Sessions
 Session 845FBEE8 (10.0.0.7:48036)=>(10.0.0.4:22) ssh SIS_OPEN
```

```
Rack1SW1#ping 10.0.0.4

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.4, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/12/48 ms
```

Note

Now check the traffic from outside to inside. Traffic to the inside should be permitted to certain ports.

```
Rack1R4#telnet 10.0.0.7 80
Trying 10.0.0.7, 80 ... Open

Rack1R4#ping 10.0.0.7

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.7, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

Note

Make sure that RIP updates are allowed across the firewall as they are multicast.

```
Rack1SW1#show ip route rip
<snip>
     150.1.0.0/24 is subnetted, 2 subnets
R       150.1.4.0 [120/1] via 10.0.0.4, 00:00:09, Vlan67
<snip>

Rack1R6#show ip inspect statistics
Packet inspection statistics [process switch:fast switch]
  tcp packets: [0:11]
   packets: [2:36]
Interfaces configured for inspection 2
Session creations since subsystem startup or last reset 5
Current session counts (estab/half-open/terminating) [0:0:0]
Maxever session counts (estab/half-open/terminating) [1:1:1]
Last session created 00:03:45
Last statistic reset never
Last session creation rate 0
Maxever session creation rate 1
Last half-open session total 0
<snip>
```

Note

Make sure the firewall can ping the connected devices thanks to the BVI configuration. This is because this traffic is not subject to the checks by the ACL applied to the firewall bridged interfaces.

```
Rack1R6#ping 10.0.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
```

Note

You may visualize L2 inspection process using the below debugging command. The output below shows the inspection process for the ICMP echo and echo-reply packets sent across the firewall.

```
Rack1R6#debug ip inspect l2-transparent packets
INSPECT L2 firewall debugging is on

Rack1SW1#ping 10.0.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/8/12 ms

RTBAP: Check AuthProxy is configured on idb=FastEthernet0/1.23 path=1
linktype=38
L2FW:Input ACL not configured or the ACL is bypassed
L2FW:Output ACL is not configured or ACL is bypassed
L2FW*:insp_l2_fast_inspection returning INSP_L2_OK
TBAP: Check if AP return traffic (fast path) foroutput
idb=FastEthernet0/0.67 IP->prot=1

L2FW*:insp_l2_fast_inspection: pak 83A7BB7C, input-interface
FastEthernet0/0.67, output-interface FastEthernet0/1.146
L2FW*:Src 10.0.0.7 dst 10.0.0.1 protocol icmp
TBAP: Check AuthProxy is configured on idb=FastEthernet0/0.67 path=1
linktype=38
```

11.43 ZFW-Based IOS Transparent Firewall

- Modify the previous scenario to use CPL for the firewall configuration.

Configuration

Note

It is possible to use CPL (Cisco Policy Language) and define transparent firewall rules using ZFW syntax. In fact, this approach has many advantages, as it operates with a better abstraction (zone) instead of interfaces and allows for easy to understand and concise configurations.

To configure ZFW, simply apply zone names to the interfaces configured for Ethernet bridging. It is possible, though not recommended to configure L3 and L2 interfaces in the same zone. However, in most cases you may want to keep those domains separate.

Any BVI interfaces on the router automatically belong to the “self” zone, and the policy should be configured appropriately. You may still use access-list on the BVIs, if there is no security zone assigned. If you want to treat the whole L2 domain behind the BVI as a single L3 security zone, you may simply assign a zone to the BVI, but not the bridging interfaces.

As for multicast and broadcast traffic types – both are not inspected by ZFW. Moreover, in transparent mode both traffic types are permitted across the firewall configured with zones. Keep this in mind when deploying ZFW configurations.

R6:
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Traffic-classes definition
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
! Inside protocols
!
class-map type inspect match-any CMAP_PROTOCOLS_FROM_INSIDE
 match protocol http
 match protocol ftp
 match protocol dns
 match protocol ssh
 match protocol icmp
```

```
!
! Outside protocols
!
class-map type inspect match-any CMAP_PROTOCOLS_TO_INSIDE
 match protocol ftp
 match protocol http
 match protocol dns
!
!
class-map type inspect CMAP_RIP_TRAFFIC
 match access-group name ACL_RIP_TRAFFIC

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Policies definition
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

policy-map type inspect PMAP_INSIDE_TO_OUTSIDE
 class CMAP_PROTOCOLS_FROM_INSIDE
  inspect
!
policy-map type inspect PMAP_OUTSIDE_TO_INSIDE
 class CMAP_PROTOCOLS_TO_INSIDE
  inspect

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Security zones and zone-pairs
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

zone security INSIDE
zone security OUTSIDE

zone-pair security ZP_INSIDE_TO_OUTSIDE source INSIDE destination
OUTSIDE
 service-policy type inspect PMAP_INSIDE_TO_OUTSIDE
!
zone-pair security ZP_OUTSIDE_TO_INSIDE source OUTSIDE destination
INSIDE
 service-policy type inspect PMAP_OUTSIDE_TO_INSIDE
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Remove the classic configs
! and apply the zone-based
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!
! Inside Interface
!
interface FastEthernet 0/0.67
 bridge-group 1
 no ip inspect INSIDE_PROTOCOLS in
 zone-member security INSIDE
 bridge-group 1 input-type-list 201

!
! Outside Interface
!
interface FastEthernet 0/1.146
 bridge-group 1
 no ip access-group OUTSIDE_IN in
 bridge-group 1 input-type-list 201
 zone-member security OUTSIDE
```

Verification

Note

You may apply the same verification techniques as were used in the previous task. First, make sure SW1 sees RIP updates from R4 as multicast packets are not subject to Transparent ZFW checks. Next, initiate a session from SW1 to R4 and check that it's being inspected by the firewall. Use the same approach you used in the previous scenario.

```
Rack1SW1#show ip route rip
<snip>
     150.1.0.0/24 is subnetted, 2 subnets
R       150.1.4.0 [120/1] via 10.0.0.2, 00:00:10, Vlan67

Rack1SW1#telnet 10.0.0.4 22
Trying 10.0.0.4, 22 ... Open
SSH-1.99-Cisco-1.25


Rack1R3#show policy-map type inspect zone-pair ZP_INSIDE_TO_OUTSIDE
sessions
 Zone-pair: ZP_INSIDE_TO_OUTSIDE

  Service-policy inspect : PMAP_INSIDE_TO_OUTSIDE

    Class-map: CMAP_PROTOCOLS_FROM_INSIDE (match-any)
      Match: protocol http
        0 packets, 0 bytes
        30 second rate 0 bps
      Match: protocol ftp
        0 packets, 0 bytes
        30 second rate 0 bps
      Match: protocol dns
        0 packets, 0 bytes
        30 second rate 0 bps
      Match: protocol ssh
        1 packets, 24 bytes
        30 second rate 0 bps
      Match: protocol icmp
        0 packets, 0 bytes
        30 second rate 0 bps
      Inspect
        Established Sessions
         Session 845FBC20 (10.0.0.7:30775)=>(10.0.0.4:22) ssh SIS_OPEN
          Created 00:00:03, Last heard 00:00:03
          Bytes sent (initiator:responder) [0:20]

<snip>
```

11.44 IOS IPS

- Configure R4 to deny inline the attacker that attempts to use IP loose source route option.
- For testing purposes, enable the ICMP echo signature and configure the IPS to notify syslog.

Configuration

Note

Historically, IOS IPS started its evolution since IOS 12.0(5)T, when a limited subset of Cisco IDS v3.x was introduced in IOS. The configuration syntax was based on the **`ip audit`** command, and IOS implemented a fixed number of IDS signatures (initially 59, but then 42 more signatures were added in IOS 12.2T). Cisco IPS used proprietary Post Office Protocol (POP) for event communication with IDS Director (part of Cisco IDS system). The configuration only supported two signature categories: “attack” and “info”. The same feature is still available in PIX and ASA platforms, though supporting different signature sets.

With IOS release 12.3(8)T Cisco introduced IDS code 4.x into IOS and named the feature as IPS, replacing **`ip audit`** command with **`ip ips`** syntax. The IPS featured support for SDEE (Security Device Event Exchange) protocol instead of legacy POP and implemented IDS 4.x signature engines, such as ATOMIC.IP, ATOMIC.ICMP, SERVICE.HTTP, SERVICE.FTP, OTHER and so on. A total of 132 built-in signatures were supported in Cisco IPS. However, now in addition to the built-in signatures, it was possible to load an external SDF (Signature definition file, and XML file used IDS 4.x format) with additional patterns that were dynamically compiled in the IPS core. This allowed for some extensibility, though the number of extensions was limited. Theoretically, it was possible to create a custom signature using XML syntax, though Cisco never openly documented it. The total number of signatures available to IOS was significantly less compared to the list available in the hardware appliances.

Recently, with the IOS 12.4(11)T the support for IPS 5.x signatures has been added to IOS. More than that, there are no more hardcoded signatures in the IOS code itself. New v5.x signature packages (in format different from the old 4.x) are now required to provide IOS with signature for the IPS engine. The packages contain the same list of signatures that a normal v5.x hardware appliance uses, and which is really large. A router is supposed to load and compile the new signatures dynamically. The configuration for the IPS subsystem is now stored separately, in the flash memory of the router, and the whole configuration process is now very similar to the hardware IPS CLI-based configuration. However, there are serious limitations in functionality, such as inability to create custom signatures.

Here are the configuration steps to enable Cisco IPS functionality using v5.x signatures:

Step 1. Downloading IOS IPS files
Step 2. Creating IOS IPS configuration directory on flash
Step 3. Configuring IOS IPS crypto key
Step 4. Enabling IOS IPS
Step 5. Loading IOS IPS signature package to router
Step 6: Tuning Signatures

Downloading IOS IPS files.

Although this step is already performed for you in the lab environment, in real life scenarios you need to access the Security/IPS section of CCO and download the following files:

IOS-Sxxx-CLI.pkg: Signature package – make sure you downloaded the latest signature package. Here xxx is the package version.

realm-cisco.pub.key.txt: This is Cisco's public-key, used to sign the signature package. The router will use this key to verify the package integrity.

Creating IOS IPS configuration directory on flash.

As mentioned previously, the new IPS code store all configuration in special files located in the flash memory. It is recommended creating a separate directory in the flash memory to store the files. Use the **mkdir** command to accomplish this. The files stored in the flash memory are compressed and not editable. Here is a quick overview of the files (replace "router" with your router's hostname):

router-sigdef-default.xml - contains all the factory default signature definitions.

router-sigdef-delta.xml - contains signature definitions that have been changed from the default.

router-sigdef-typedef.xml - is a file that has all the signature parameter definitions.

router-sigdef-category.xml - has all the signature category information, such as category ios_ips basic and advanced.

`router-seap-delta.xml` - contains changes made to the default SEAP parameters. SEAP is the signature event processing engine, similar to the one found in the hardware appliance.

`router-seap-typedef.xml` - contains all the SEAP parameter definitions

`Configuring IOS IPS crypto key.`

Open the file **`realm-cisco.pub.key.txt`** in a text editor or list it content using the **`more`** command if the file is located in the flash memory. Copy the contents of the file, enter the configuration mode using the command **`configure terminal`** and paste the file content. This will import the public key into the routers configuration. Don't forget to save the running configuration after this.

`Enabling IOS IPS`.

This step comprises IPS initialization, creating an inspection rule, specifying the configuration file location and retiring unneeded signature categories. The first thing you need to do is to configure basic IPS settings:

```
ip ips name <NAME> [list <ACL_NAME>]
ip ips config location flash:<directory>
ip ips notify sdee
ip ips notify log
```

The first two lines create the IPS rule – you may associate and access-list with the IPS rule, to limit the scope of inspection traffic. The second line specifies the flash directory you created previously, to store the IPS configuration. The last two commands instruct the IPS to notify listeners connecting via SDEE for IPS monitoring and send IPS events into system log.

The next important step consists of retiring most IPS signatures. The issue is that the IPS package is so big, that the IOS will not be able to fit all signatures into memory, unless there is plenty of it (above a gigabyte). Thus, you need to enter the IPS configuration mode and retire most, if not all signatures initially. Retired signatures are not compiled in the memory; disabled signatures are compiled but not triggered. Thus it is important to retire and not simply disable the signatures. Use the following syntax:

```
ip ips signature-category
 category all
  retired true
  exit
 category ios_ips basic
  retired false
  exit
 exit
```

The above configuration will retire signatures bases on categories (logical grouping). Optionally, you may retire the all category and then selectively enable some signatures. Notice the use of the **exit** command – this syntax is the same as when configuring an IPS 5.x appliance via CLI.

The last thing you need to do in order to activate the IPS is apply the IPS rule to an interface. This will make the configuration active, even though there are no signatures loaded yet.

```
interface FastEthernet 0/0
  ip ips <NAME> in
```

You may choose either in or out directions to inspect ingress or egress traffic. Commonly, you need to inspect in both directions on the same interface, but you may apply the same rule to different interface in the same direction to have the same effect.

Loading IOS IPS signature package to router.

This step completes the IPS initialization. The signatures are loaded using special destination know as IDCONF. In order to complete the signature load, use the command **copy <URL> idconf**, e.g.

copy ftp://cisco:cisco@10.0.0.100/IOS-S310-CLI.pkg idconf

or

copy flash://IOS-S310-CLI.pkg idconf

This will initiate the compile process in the router memory. All retired signatures are ignored during the process. When the process is finished, use the command **show ip ips signature count** to check the statistics on the loaded signatures.

Tuning Signatures.

Basically, what you could do with the signatures is enabling or disabling them, changing event-action, alarm-severity and fidelity-rating attributes. You cannot tune the engine settings, not create custom signatures. Here is a sample list of steps that you need to do in order to tune a signature:

```
ip ips signature-definition
  signature 2004 0
    status
      enabled true
      retired false
    exit
    engine
      event-action produce-alert
    exit
    alert-severity high
    fidelity-rating 100
  exit
exit
```

The above sample shows all the basic settings you can change for a signature. Additionally, you may operate with signature categories, like we did when first loading the signature file. You may enable/disable/retire/unretired groups of signatures without having to deal with every one of them individually. Additionally, you may change the action for all signatures in a group at once, or change severify/fidelity.

```
ip ips signature-category
  category attack
    retired false
    event-action deny-attacker-inline
    enabled true
    exit
  exit
```

The above configuration will unretire/enable all “attack” signatures and specify a deny action for all of them. If you having hard time finding the right signature, use the IDM interface to lookup signature and/or signature categories based on name, ID, engine and so, as it’s hard to navigate through the IOS IPS configuration.

The last feature supported by IOS IPS, is a limited SEAP (Signature Event Action Processing) feature. Specifically, you may assign Target Value Rating to IP subnets using the following syntax:

```
ip ips event-action-rules
  target-value medium target-address 150.1.2.0/24
  exit
```

This will affect the risk-rating computation formula for attacks towards this IP subnet. However, there are no event-filter or event-overrides implemented in the IOS IPS yet, so those setting are only informational.

R4:

Note

Perform the steps required to initialize the IPS prior to starting the configuration part.

```
ip ips name MYIPS
ip ips config location flash:ips
ip ips notify sdee
ip ips notify log
!
interface FastEthernet 0/1
 ip ips MYIPS in
!
interface FastEthernet 0/0
 ip ips MYIPS in
!
ip ips signature-category
  category all
    retired true
    exit
  exit
!
```

```
ip ips signature-definition
  signature 2004 0
    status
      enabled true
      retired false
    exit
  exit
  signature 1004 0
   status
    enabled true
    retired false
    exit
   engine
     event-action deny-attacker-inline
     exit
  exit
```

Verification

Note

Start by checking the overall IPS configuration. After this, initiate ping across R6 and check R4's syslog for events:

```
Rack1R4#show ip ips all

IPS Signature File Configuration Status
    Configured Config Locations: flash:ips/
    Last signature default load time: 23:14:30 UTC Jun 27 2009
    Last signature delta load time: 23:21:59 UTC Jun 27 2009
    Last event action (SEAP) load time: -none-

    General SEAP Config:
    Global Deny Timeout: 3 seconds
    Global Overrides Status: Enabled
    Global Filters Status: Enabled

IPS Auto Update is not currently configured

IPS Syslog and SDEE Notification Status
    Event notification through syslog is enabled
    Event notification through SDEE is enabled

IPS Signature Status
    Total Active Signatures: 2
    Total Inactive Signatures: 2318

IPS Packet Scanning and Interface Status
    IPS Rule Configuration
      IPS name MYIPS
    IPS fail closed is disabled
    IPS deny-action ips-interface is false
    Interface Configuration
      Interface FastEthernet0/0
        Inbound IPS rule is MYIPS
        Outgoing IPS rule is not set
      Interface FastEthernet0/1
        Inbound IPS rule is MYIPS
        Outgoing IPS rule is not set

IPS Category CLI Configuration:
    Category all:
        Retire: True
```

```
Rack1R1#ping 204.12.1.254

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 150.1.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms

%IPS-4-SIGNATURE: Sig:2004 Subsig:0 Sev:25 ICMP Echo Request
[155.1.146.1:8 -> 204.12.1.254:0] VRF:NONE RiskRating:25
%IPS-4-SIGNATURE: Sig:2004 Subsig:0 Sev:25 ICMP Echo Request
[155.1.146.1:8 -> 204.12.1.254:0] VRF:NONE RiskRating:25
%IPS-4-SIGNATURE: Sig:2004 Subsig:0 Sev:25 ICMP Echo Request
[155.1.146.1:8 -> 204.12.1.254:0] VRF:NONE RiskRating:25
%IPS-4-SIGNATURE: Sig:2004 Subsig:0 Sev:25 ICMP Echo Request
[155.1.146.1:8 -> 204.12.1.254:0] VRF:NONE RiskRating:25
%IPS-4-SIGNATURE: Sig:2004 Subsig:0 Sev:25 ICMP Echo Request
[155.1.146.1:8 -> 204.12.1.254:0] VRF:NONE RiskRating:25
```