- Switch and VLAN Security -

Switch Port Security

Port Security adds an additional layer of security to the switching network.

The MAC address of a host generally does not change. If a specific host will always remain connected to a specific switch port, then the switch can filter all *other* MAC addresses on that port using Port Security.

Port Security supports both **statically** mapping MAC addresses, and **dynamically** learning addresses from traffic sent on the port.

To enable Port Security on an interface:

Switch(config)# *interface gi1/10*
Switch(config-if)# *switchport port-security*

By default, Port Security will allow **only one MAC** on an interface. To adjust the maximum number of allowed VLANs, up to 1024:

Switch(config-if)# *switchport port-security maximum 2*

To statically map the allowed MAC addresses on an interface:

Switch(config-if)# *switchport port-security mac-address 0001.1111.2222*
Switch(config-if)# *switchport port-security mac-address 0001.3333.5555*

Only hosts configured with the above two MAC addresses will be allowed to send traffic through this port.

If the *maximum* number of MAC addresses for this port had been set to *10*, but only two were statically mapped, the switch would dynamically learn the remaining eight MAC addresses.

Port Security refers to dynamically learned MAC addresses as **sticky addresses.** Sticky addresses can be *aged* out after a period of inactivity, measured in minutes:

Switch(config-if)# *switchport port-security aging time 10*

Port Security aging is *disabled* by default.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

Switch Port Security (continued)

A **violation** occurs if an *unauthorized* MAC address attempts to forward traffic through a port. There are three violation **actions** a switch can perform:

- **Shutdown –** If a violation occurs, the interface is placed in an *errdisable* state. The interface will stop forwarding *all* traffic, including non-violating traffic, until it is removed from an errdisable state. This is the **default** action for Port Security.
- **Restrict –** If a violation occurs, the interface will remain online. Legitimate traffic will be forwarded, and unauthorized traffic will be dropped. Violations are **logged**, either via a syslog message or SNMP trap.
- **Protect –** If a violation occurs, the interface will remain online. Legitimate traffic will be forwarded and unauthorized traffic will be dropped, but *no* logging will occur.

To configure the desired Port Security violation action:

Switch(config-if)# *switchport port-security violation shutdown*
Switch(config-if)# *switchport port-security violation restrict*
Switch(config-if)# *switchport port-security violation protect*

To view Port Security configuration and status for a specific interface:

Switch# *show port-security interface gi1/10*

```
Port Security: Enabled
Port status: SecureUp
Violation mode: Shutdown
Maximum MAC Addresses: 10
Total MAC Addresses: 10
Configured MAC Addresses: 2
Aging time: 10 mins
Aging type: Inactivity
SecureStatic address aging: Enabled
Security Violation count: 0
```

Note that the *maximum MAC addresses* is set to *10,* and that the *total MAC addresses* is currently at *10* as well. The *violation mode* is set to *shutdown.*

If another MAC address attempts to forward traffic through this port, the port will be place in an errdisable state.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

802.1x Port Authentication

802.1x Port Authentication forces a host device to *authenticate* with the switch, before the switch will forward traffic on behalf of that host. This is accomplished using the **Extensible Authentication Protocol over LANs (EAPOL).** 802.1x only supports **RADIUS** servers to provide authentication.

Both the switch *and* the host must support 802.1x to use port authentication:

- If the *host* supports 802.1x, but the switch does not – the host will not utilize 802.1x and will communicate normally with the switch.
- If the *switch* supports 802.1x, but the host does not – the interface will stay in an **unauthorized** state, and will not forward traffic.

A switch interface configured for 802.1x authentication stays in an **unauthorized** state until a client successfully authenticates. The only traffic permitted through an interface in an unauthorized state is as follows:

- EAPOL, for client authentication
- Spanning Tree Protocol (STP)
- Cisco Discovery Protocol (CDP)

To globally enable 802.1x authentication on the switch:

Switch(config)# *dot1x system-auth-control*

To specify the authenticating RADIUS servers, and configure 802.1x to utilize those RADIUS servers:

Switch(config)# *aaa new-model*
Switch(config)# *radius-server host 192.168.1.42 key CISCO*
Switch(config)# *aaa authentication dot1x default group radius*

Finally, 802.1x authentication must be configured on the desired interfaces:

Switch(config)# *interface gi1/10*
Switch(config-if)# *dot1x port-control auto*

An interface can be configured in one of three 802.1x states:

- **force-authorized –** The interface will *always* authorize any client, essentially disabling authentication. This is the **default** state.
- **force-unauthorized –** The interface will *never* authorize any client, essentially preventing traffic from being forwarded.
- **auto –** The interface will actively attempt to authenticate the client.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

VLAN Access-Lists

Normally, access-lists are applied to interfaces to filter traffic *between* networks or VLANs.

VLAN Access-Lists (VACLs) filter traffic *within* a VLAN. Because *intra-VLAN* traffic does not traverse an interface, it cannot be directly filtered using a normal access-list.

For a VACL to function, traffic must first be *identified* using an access-list:

```
Switch(config)# ip access-list extended BLOCKTHIS
Switch(config-ext-nacl)# permit ip host 10.1.5.10 10.1.0.0 0.0.255.255
```

VACLs support three types of access-lists:

- **IP**
- **IPX**
- **MAC address**

Note that the access-list is **not used to deny or permit traffic**. Instead, it **identifies traffic** for the VACL to deny or permit. The ACL *permit* parameter functions as a *true* statement, and a *deny* parameter functions as a *false* statement.

To configure the VACL itself:

```
Switch(config)# vlan access-map MY_VACL 5
Switch(config-access-map)# match ip address BLOCKTHIS
Switch(config-access-map)# action drop

Switch(config-access-map)# vlan access-map MY_VACL 10
Switch(config-access-map)# action forward

Switch(config)# vlan filter MY_VACL vlan-list 102
```

The first line creates a *vlan access-map* named *MY_VACL*. Traffic that *match*es entries in the *BLOCKTHIS* access-list will be *drop*ped.

The final *vlan access-map* entry contains only an *action* to *forward*. This will apply to **all other traffic,** as no other access-list was specified. Finally the VACL is applied to *VLAN 102*.

Notice that every *access-map* statement contains a sequence number - *5* and *10* respectively in the above example. This dictates the order in which the rules should be processed.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

Private VLANs

Private VLANs (PVLANs) allow for further segmentation of a subnet *within* a VLAN. This is accomplished by creating **secondary** VLANs within a **primary** VLAN.

The *secondary* VLAN **can communicate only** with the *primary* VLAN, and not any other secondary VLANs.

There are two types of secondary VLANs:

- **Community**
- **Isolated**

Hosts *within* a **community** VLAN can communicate with each other, and with the primary VLAN.

Hosts *within* an **isolated** VLAN *cannot* communicate with each other. However, the hosts can still communicate with the primary VLAN.

Private VLANs are only locally-significant to the switch. VTP will not pass information on Private VLANs to other switches.

Each switch port in a private VLAN must be configured with a specific *mode*:

- **Promiscuous**
- **Host**

A **promiscuous** port can communicate with the primary VLAN and all secondary VLANs. Gateway devices such as routers and switches should be configured as promiscuous ports.

A **host** port can communicate only with promiscuous ports, and other ports within the *local* community VLAN. Host devices such as workstations should be configured as host ports.

Private VLANs allow a group of hosts to be segmented within a VLAN, while still allowing those devices to reach external networks via a promiscuous gateway.

Service providers can use private VLANs to segregate the traffic of multiple customers, while allowing them to share a gateway.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

Private VLAN Configuration

Configuring private VLANs involves the following:

- Creating secondary VLANs
- Creating primary VLANs
- Associating secondary VLANs to a primary VLAN
- Associating ports to either a primary or secondary VLAN

To create secondary VLANs:

```
Switch(config)# vlan 100
Switch(config-vlan)# private-vlan community

Switch(config)# vlan 101
Switch(config-vlan)# private-vlan isolated
```

To create a primary VLAN, and associate secondary VLANs:

```
Switch(config)# vlan 50
Switch(config-vlan)# private-vlan primary
Switch(config-vlan)# private-vlan association 100,101
```

To associate a port to a primary and secondary VLAN:

```
Switch(config)# interface gi1/10
Switch(config-if)# switchport private-vlan host
Switch(config-if)# switchport private-vlan host-association 50 101
```

The *gi1/10* port has been identified as a *host* port, and associated with primary VLAN *50,* and secondary VLAN *101*.

To configure a promiscuous port:

```
Switch(config)# interface gi1/20
Switch(config-if)# switchport private-vlan promiscuous
Switch(config-if)# switchport private-vlan mapping  50 100.101
```

The promiscuous *gi1/20* port is associated with primary VLAN *50,* and the two secondary VLANs, *100* and *101*.

To allow the primary VLAN SVI to perform Layer-3 forwarding for specified secondary VLANs:

```
Switch(config)# interface vlan 50
Switch(config-if)# private-vlan mapping 100,101
```

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

DHCP Snooping

Dynamic Host Control Protocol (DHCP) provides administrators with a mechanism to *dynamically* assign IP addresses, rather than manually configuring the address on each host.

DHCP servers **lease** out IP addresses to DHCP clients, for a specific period of time. There are four steps to this DHCP process:

- When a DHCP client first boots up, it broadcasts a **DHCPDiscover** message, searching for a DHCP server.
- If a DHCP server exists on the local segment, it will respond with a **DHCPOffer,** containing the offered IP address, subnet mask, etc.
- Once the client receives the offer, it will respond with a **DHCPRequest**, indicating that it will accept the offered protocol information.
- Finally, the server responds with a **DHCPACK**, acknowledging the clients acceptance of offered protocol information.

Malicious attackers can place a rogue DHCP server on the trusted network, intercepting DHCP packets while masquerading as a legitimate DHCP server. This is a form of **spoofing** attack, which intends to gain unauthorized access or steal information by sourcing packets from a *trusted* source. This is also referred to as a *man-in-the-middle* attack.

DHCP attacks of this sort can be mitigated by using **DHCP Snooping.** Only specified interfaces will accept DHCPOffer packets – unauthorized interfaces will discard these packets, and then place the interface in an errdisable state.

DHCP Snooping must first be globally enabled on the switch:

```
Switch(config)#  ip dhcp snooping
```

Then, DHCP snooping must be enabled for one or more VLANs:

```
Switch(config)#  ip dhcp snooping vlan 5
```

By default, all interfaces are considered *untrusted* by DHCP Snooping. Interfaces connecting to legitimate DHCP servers must be *trusted:*

```
Switch(config)#  interface gi1/20
Switch(config)#  ip dhcp snooping trust
```

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com), unless otherwise noted. All other material copyright © of their respective owners. This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.

Dynamic ARP Inspection

Another common man-in-the-middle attack is **ARP spoofing,** sometimes referred to as **ARP poisoning**. A malicious host can masquerade as another host, by intercepting ARP requests and responding with its *own* MAC address.

Dynamic ARP Inspection (DAI) mitigates the risk of ARP Spoofing, by inspecting all ARP traffic on *untrusted* ports. DAI will confirm that a legitimate MAC-to-IP translation has occurred, by comparing it against a trusted database.

This MAC-to-IP database can be statically configured, or DAI can utilize the DHCP Snooping table, assuming that DHCP Snooping has been enabled.

To globally enable DAI for one or more VLANs:

Switch(config)# *ip arp inspection vlan 100*

By default, all interfaces in VLAN 100 will be considered **untrusted**, and thus subject to inspection by DAI. Trunk ports to other switches should be configured as **trusted**, indicating no inspection will occur, as each switch should handle DAI locally:

Switch(config)# *interface gi1/24*
Switch(config-if)# *ip arp inspection trust*

To create a manual MAC-to-IP database for DAI to reference:

Switch(config)# *arp access-list DAI_LIST*
Switch(config-acl)# *permit ip host 10.1.1.5 mac host 000a.1111.2222*
Switch(config-acl)# *permit ip host 10.1.1.6 mac host 000b.3333.4444*

Switch(config)# *ip arp inspection filter DAI_LIST vlan 100*

If an ARP response does not match the MAC-to-IP entry for a particular IP address, then DAI drops the ARP response and generates a log message.

* * *

All original material copyright © 2014 by Aaron Balchunas (aaron@routeralley.com),
unless otherwise noted. All other material copyright © of their respective owners.
This material may be copied and used freely, but may not be altered or sold without the expressed written consent of the owner of the above copyright. Updated material may be found at http://www.routeralley.com.