Contents

FHS: First Hop Security

FHS pocket guide:intro

There are two goals of this document:

- a short overview of each of the Cisco First Hop Security features and their interactions
- an easy to reuse template configuration, built step-by-step by layering the features together.

Before you begin to plan to deploy Cisco IPv6 FHS we recommend the reader to ask themselves a simple question: "If I also have IPv4 deployed, am I using the IPv4 SISF features (like ARP guard and DHCP guard) ?" If the answer to this question is "NO" - then, the most basic IPv6 FHS feature (RA Guard) will be all that you need to achieve functional parity between IPv4 and IPv6, from the FHS point of view.

The attack vectors presented here are all pretty similar in IPv4. In that case, the network that only protects the IPv6 stack resembles a property with a reinforced locked door with all of the windows open all the time ! Simply stated if you have come here worried about First Hop Security in IPv6 - then it is assumed that you have also closed the same vulnerabilities in the IPv4 plane too.

However, there is a point to be made about the above logic and the interaction of IPv4 and IPv6. Unprotected IPv6 may result in leaking the information when using IPv4-only VPN , so this needs to be taken into consideration.

NOTE: At this time we do not cover all of the FHS features, but we plan to extend the document to add more material. Please let us know what you think of this way of presenting the information to you.

FHS configuration principles

Most FHS features are configured in a two-step fashion: firstly you define a policy which describes the behaviour of the feature, secondly you apply this policy to a "domain" (being the box, vlan, or interface). Different policies that define different behaviours can be applied to different intersecting domains. The decision to use a specific policy is taken by the most specific domain to which the policy is applied.

Note however that not all configuration domains are supported on all platforms.

RA guard

RA guard use-case

IPv6 Routers send a special packet known as a "Router Advertisement" to all hosts on all connected segments. These packets contain the basic configuration information necessary to bring up IPv6 communications on their attached hosts. The base RA protocol in RFC4861 is unauthenticated. The consequence of this is that the host will accept any Router Advertisement. This fact opens up the possibility for an accidental or malicious rogue RA being sent with the impact of misconfiguring the IPv6 stack on the host.

RA Guard is a feature that allows the operator of a Layer 2 switch to pre-determine which switch ports are actually router facing. In that case Router Advertisements received on any other ports cannot be valid and are dropped by the switch, and hence they never reach other nodes on the same link. RA guard can also validate the source of the RA, the prefix list, the preference and any other information carried within it. It can validate the cryptographic credentials when provided (as defined in Secure Neighbor Discovery specification, i.e. SeND) to provide nodes that don?t support SeND with a level of security equivalent to those that do support it.

From a deployment perspective, RA guard should be deployed on the the first hop L2 switch where end points will access the network. Applying RA guard at this point in the network ensures that the RA does not propagate to other end systems in the network. RA guard will work on switch access ports and trunk ports. The only consideration to take into account is where valid RA?s will come from.

RA guard is specified in RFC6105.

You can watch a 5-minute video describing the scenario and the risk this feature mitigates, on YouTube.

RA guard sample configurations

A simple configuration:

```
interface Ethernet0/1
  ipv6 nd raguard
```

This configuration will drop any RAs received on this interface thus enforcing the policy that this port only connects to the end hosts. This configuration will apply to the vast majority of ports that will provide access to end systems. It is very simple, but not very flexible. The full RA guard feature allows much more granularity than simply the act of dropping RAs on specific ports.

Below we show a configuration that will permit RAs only if they have the M and O bits set, and enforce that the subsequent DHCP advertised prefix is within the company's range of 2001:db8:cafe::/48 :

```
!
ipv6 nd raguard policy ONLY-DHCPv6-RAs
  ! role 'router' allows the RAs through but triggers deep inspection
   device-role router
  ! The RAs that we let through have to have Managed flag set.
   managed-config-flag on
  ! The Other configuration flag also needs to be set.
   other-config-flag on
  ! Only allow the RAs that advertise the prefixes from our own address space
  match ra prefix-list IPv6-SPACE
!
! . . .
!
interface Ethernet0/0
   description connection to R1 from Sw3
   switchport
   switchport access vlan 100
   switchport mode access
   ! Attach the policy to the port connecting to the router
   ipv6 nd raguard attach-policy ONLY-DHCPv6-RAs
   spanning-tree portfast
!
! . . .
!
ipv6 prefix-list IPv6-SPACE permit 2001:db8:cafe::/48 ge 64 le 64
```

There is also an excellent document with sample configurations at Support Forums for more information on RA Guard.

Logging dropped packets

It is useful to be able to see which packets are being dropped by FHS. However note that by default, packets are dropped silently. In order to see the packets are being dropped, please add this line to your configuration:

```
!
ipv6 snooping logging packet drop
!
```

This configuration will cause syslog messages to be written by the FHS components whenever a packet is dropped. For example here is a syslog message indicating that RA Guard has dropped an RA on a particular port:

```
*Jul 27 17:34:54.953: %SISF-4-PAK_DROP: Message dropped A=FE80::A8BB:CCFF:FE00:6900 G=- V=100 I=E
```

RA guard evasion

By default the RA guard filter process acts on well-formed packets.

There is the possibility to bypass the filter by sending specially crafted packets (such as one needing fragmentation). These attacks have been publicly documented in IETF draft gont-v6ops-ra-guard-evasion

If you need to protect from a malicious attacker on the first hop link using these sorts of attacks then additional configuration is needed.

The reason for this is in general RA guard assumes well-formed IPv6 packets which are not fragmented and not obscured by additional headers. Thus, if the malicious party tries to obscure the RA behind fragments, it may go unnoticed through the port and affect the other hosts upon reassembly. This additional configuration is aimed at filtering this kind of packets.

Note: the configuration below involves a trade-off between IPv6 standards compliance and the attack vector protection. Use it only after careful evaluation. ACL configuration may be also platform-dependent (see the end of this article).

Base IPv6 specification does not imply the filtering of the IPv6 fragments - thus, any protocols that may employ fragments will get impacted. Some examples of this could be SeND, OSPFv3, Kerberos. However, on an access port one usually does not these protocols, and the network role of a host on such a port is usually client - so the impact will likely be small.

The configuration blocking the fragments is very conservative, because in order to be potentially affected, the hosts have to:

NOT implement the recommendations from RFC6980 AND allow overlapping fragments. A lot of modern operating systems do not allow that.

So, from the practical standpoint, a best compromise could be a more relaxed configuration permitting the fragments while filtering the RAs with RA guard and denying the undetermined-transport.

```
!
interface GigabitEthernet1/0/1
 ipv6 traffic-filter nofrags in
!
ipv6 access-list nofrags
 !!!! the line below may be uncommented for the most
 !!!! conservative environments, after studying the implications
 ! deny ipv6 any FE80::/64 fragments
 deny ipv6 any FE80::/64 undetermined-transport
 permit ipv6 any any
!
```

IPv6 snooping

IPv6 snooping, (also known as "binding integrity guard") is actually a "bundled" feature that combines:

- RA guard / DHCP guard
- IPv6 address gleaning
- IPv6 ND inspection

Since IPv6 snooping included the guard functions, if you enable IPv6 snooping, you do not need to explicitly configure RA guard / DHCP guard on the same port.

Simply configuring IPv6 snooping should be valid for most situations, however, if a more advanced configuration for RA guard is needed, then a specific RA guard policy will need to be configured as shown in the RA guard configuration section.

The below configuration shows a default configuration for IPv6 snooping.

```
!
ipv6 snooping policy HOST
!
!
interface GigabitEthernet1/0/2
 switchport access vlan 201
 switchport mode access
 ipv6 snooping attach-policy HOST
!
```

To verify the snooping policy you can use the ?show ipv6 snooping policy [policy name]? command as shown below.

```
FirstHopSwitch#show ipv6 snooping policy HOST
Policy HOST configuration:
  security-level guard                  <-  RA and DHCP messages are dropped
  device-role node                      <-  Only end points are expected on this port
  protocol ndp                          <-  Gleaning addresses from ND process
  protocol dhcp                         <-  Gleaning addresses from DHCP
Policy host is applied on the following targets:
Target               Type  Policy               Feature        Target range
Gi1/0/2              PORT  HOST                 Snooping       vlan all
```

Note that when IPv6 snooping is configured the default security is set to "guard". This setting can cause problems when the snooping policy is attached to a port that has a DHCPv6 server connected to it. This policy setting will drop the DHCPv6 messages. To allow the DHCPv6 messages through, the port must be configured as a trusted port or a specific DHCPv6 guard policy must be configured.

For a short demonstration of IPv6 Snooping in action, take a look at this YouTube video.

IPv6 address gleaning

Address gleaning learns the IPv6 addresses of devices connected to that link and is a prerequisite for more advanced FHS features like Source-Guard. The learning is done by examining information in the ND and DHCPv6 packets, (in particular the addresses carried in them). However, by default the DHCPv6 server messages are dropped - so in order to glean from DHCPv6 messages the guard policy must be applied on the port connecting the valid DHCPv6 server, that allows the DHCPv6 messages.

The FHS code learns the addresses and installs them into the binding table. Each entry contains the source the address was learned from, the address itself, the MAC address, interface, vlan, priority level, age, state and time left.

IPv6 ND inspection

ND inspection verifies the sanity of the ND messages that pass through the device. It can also enforce limits on the number of the addresses per port. This feature enforces the ND process by ensuring that all parties are stepping through all the correct steps in the ND process. The ND inspection process builds the neighbor binding table.

Verifying the binding table

In the below example, there are two entries in the binding table, both learned via ND, belonging to the same device connected on the interface Ethernet2/0 in vlan100 - one address is link-local, and one is global. We can see both entries are in the REACHABLE state and will stay so for another almost 300 seconds.

```
Sw2#show ipv6 neighbors binding
Binding Table has 2 entries, 2 dynamic
Codes: L - Local, S - Static, ND - Neighbor Discovery, DH - DHCP, PKT - Other Packet, API - API c
Preflevel flags (prlvl):
0001:MAC and LLA match     0002:Orig trunk            0004:Orig access
0008:Orig trusted trunk    0010:Orig trusted access   0020:DHCP assigned
0040:Cga authenticated     0080:Cert authenticated    0100:Statically assigned


    IPv6 address                            Link-Layer addr Interface vlan prlvl  age   state
ND  FE80::A8BB:CCFF:FE00:6900               AABB.CC00.6900  Et2/0      100  0005   11s  REACHABLE
ND  2001:DB8:23::2                          AABB.CC00.6900  Et2/0      100  0005   10s  REACHABLE

Sw2#
```

IPv6 data gleaning

Sometimes just monitoring DHCPv6 and ND flows is not enough to learn addresses (for example, the switch might have just rebooted, or the DAD NS might have been lost en-route to the switch, and the host already started to send the data). In order to still maintain the correct representation of the attached addresses, the switch can punt data packets from unknown sources on the link to the address gleaning code, which will then send a DAD Neighbor Solicitation for the address being verified.

If the host indeed has the IPv6 address, it will reply with the Neighbor Advertisement. This will serve as the first-come-first-serve proof of ownership, and therefore the new entry with the address will be installed in the binding table.

This option is fairly CPU intensive and therefore is off by default. To enable it in the configuration, configure the "data-glean" keyword under the snooping policy, like the example below:

```
ipv6 snooping policy FOO
  data-glean
```

This feature may not be implemented on all the platforms at this time, so please verify that you can apply the policy with data-glean after configuring it.

IPv6 device tracking

Device Tracking is an extension to the basic address gleaning process. With Device Tracking the switch maintains the currency of the gleaned address entries. Stale entries are re-verified using the NS-NA DAD exchange similar to the initial ownership verification, and cleaned up upon failure.

This means that the binding table contains a reasonably accurate representation of the addresses on the link, and can be used as an authoritative source of policy for other features, like source guard.

Device Tracking is off by default and is configured under the IPv6 snooping policy.

```
ipv6 snooping policy FOO
  tracking enable
```

These features do require CPU time to generate and process the ND traffic required for these features to work. These features should only be deployed in situations where very granular information is needed about end systems. These features should not be deployed on platforms that are aggregating large numbers of end systems as the ND operations can overwhelm the CPU. Further analysis should be done when applying the data glean and device tracking FHS features to ensure that these features are needed where they will be deployed and that the platform is capable of handling the load.

IPv6 snooping logging

The volatility of IPv6 addresses presents a particular challenge to network administrators: how to understand what is happening over time on the first hop link. The binding table on the FHS-enabled switch presents a source of knowledge, however this information is of a snapshot nature and sometimes what is also needed is a historical overview. To help with this, one can configure the logging of IPv6 snooping. With this enabled all of the changes in the binding table will be serialised into the ordinary syslog messages which can be sent, processed, and analysed as usual.

In order to log IPv6 snooping events, use the following configuration line:

```
!
ipv6 neighbor binding logging
!
```

Note that this logging activity can generate a lot of messages if the platform is an aggregation point for a large number of end systems. This logging activity can increase the CPU if syslog messages are exported externally or overrun configured logging buffers quicker on the platform. Further analysis should be done to determine the impact of this feature when deploying.

Enforcing IPv6 address counts and binding table size limits

To ensure that a rogue host can not negatively impact the switch, if you turn on the IPv6 snooping, you MUST define the appropriate limits for the number of entries in the neighbor table.

IPv6 snooping can be used to define these limits. As shown below the IPv6 policy limits the number of addresses that can be learned on the port to 2.

```
!
ipv6 snooping policy HOST
 limit address-count 2
!
```

This policy allows for two addresses to be learned. One will be a link local address and the other address will be whatever address is learned first (SLAAC, DHCPv6, privacy, etc.) It is **imperative** when using this feature to appropriately control how addresses are assigned to end systems. For example, if a device is going to be assigned an address via DHCPv6 then SLAAC and privacy extensions must be disabled on the end

system. The administrators must keep in mind that some end systems do not allow to turn off the privacy addresses in case SLAAC is used for address assignment - therefore applying the above configuration in the SLAAC environment will cause connectivity outages that are hard to troubleshoot.

When deploying this feature, the initial deployment of the policy MUST start with a larger values for the limits to ensure that all end systems are reachable and that behaviors are well understood. As proper operations are confirmed this window size can be decreased.

The neighbor binding table size can be further restricted by the following globally configured command.

```
!
ipv6 neigh binding max-entries [max # of entries]
!
```

The same recommendations apply when using this command. Start conservatively by allowing a larger number than expected, observe the address usage trends and tweak the number appropriately. Keep in mind that IPv6 end systems will have at a minimum 2 IPv6 addresses associated with the device.

IPv6 snooping configuration example

This configuration example attaches the IPv6 snooping policy at the "box" level. This means we need to configure the RA guard explicitly on the port connecting the router, to permit the RAs from the router on all the VLANs. Note here that the configured policy on the interface has precedence over the globally configured policy.

```
ipv6 neighbor binding logging
ipv6 snooping logging packet drop
ipv6 nd raguard policy ROUTER
  device-role router
!
ipv6 snooping policy SNOOP
   tracking enable
!
ipv6 snooping attach-policy SNOOP
!
interface Ethernet2/0
 description Trunk to the router
 switchport
 switchport trunk encapsulation dot1q
 switchport mode trunk
 ipv6 nd raguard attach-policy ROUTER
```

We can verify that indeed the policies are applied as expected:

```
Sw2#sh ipv6 snooping policies
Target               Type  Policy               Feature        Target range
Et2/0                PORT  ROUTER               RA guard       vlan all
Box                  BOX   SNOOP                Snooping       vlan all
Sw2#
```

IPv6 source guard

IPv6 source guard builds on IPv6 snooping and takes the advantage of the accurate representation of the addresses on the first hop built by the address gleaning/device tracking processes. It can be used to make blind spoofing of addresses much harder for the attacker.

It achieves this by verifying the source address against the binding table: if the source address is not in the binding table, the packet is dropped (with an optional punt to the gleaning module to initiate the learning process). This means that an attacker who wants to spoof an address must also perform the full DAD cycle for it such that the address gets entered into the binding table.

You can watch a short video of Source Guard in action on YouTube.

One might think that requiring the NS-NA cycle and adding the client entry explicitly might only add load to the system, because an attacker could perform DAD for as many addresses as they wanted to. In fact this is not the case if you also configure a limit for the number of addresses that can be installed per MAC. If the attacker then spoofs the MAC address then this can be taken care of by address-family independent L2 security features, like using port security with a MAC address limit per port. See for example [1]

This is how the FHS suite of commands allows to limit the impact that the attacker might have.

Cumulative example configuration: RA guard, IPv6 snooping/logging, Source Guard

In the following example we enable RA Guard, we explicitly enable port E2/0 as a router port, we enable Source Guard and we log dropped packets

```
ipv6 nd raguard policy ROUTER
 device-role router
!
! log which packets are dropped
ipv6 snooping logging packet drop
ipv6 snooping policy SNOOP
 tracking enable reachable-lifetime 300
!
ipv6 snooping attach-policy SNOOP
!
ipv6 source-guard policy SG
!
ipv6 source-guard attach-policy SG
!
!
interface Ethernet2/0
 description Router trunk
 switchport
 switchport trunk encapsulation dot1q
 switchport mode trunk
 ipv6 nd raguard attach-policy ROUTER
!
! The below is for testing of the device tracking.
!!!!!! ipv6 neighbor binding stale-lifetime  300
!
!
! Log the changes to the binding table
ipv6 neighbor binding logging
!
```

Configuration snippets

This section contains configuration examples for the various FHS components and is intended as a reference.

RA guard: host-only port

A simple configuration enabling E0/1 as a host-only port. Any RA's received on this port will be automatically dropped.

```
   interface Ethernet0/1
     ipv6 nd raguard
```

RA guard: One router port, all the other ports host-only

A configuration where one port (E2/0) is explicitly configured as a router port and the rest of the ports are (by default) host ports. The RA-guard is turned on by default in VLAN100 only.

In this configuration the only port on which RA's will be accepted is E2/0

```
ipv6 nd raguard policy HOST
!
ipv6 nd raguard policy ROUTER
 device-role router
!
vlan configuration 100
 ipv6 nd raguard attach-policy HOST
!
interface Ethernet2/0
 description Router
 switchport
 switchport trunk encapsulation dot1q
 switchport mode trunk
 ipv6 nd raguard attach-policy ROUTER
```

Connecting a trunk to a virtualization server

A frequent scenario in today's highly virtualized environment is connecting the server with the hypervizor via a trunk port - with the respective virtual machines connected via the hypervizor's virtual switch into the respective VLANs.

Given that the Virtual Machines may be under dramatically different administrators, it can be beneficial to enforce the first hop security on the switch trunk port.

NOTE: the Etherchannel configuration is not supported at this time (The http://www.cisco.com/en/US/docs/ios-xml/ios/ipv6/configuration/15-2mt/ip6-ra-guard.html#GUID-589AF00C-7499-439F says it is but it is a documentation bug to be fixed.)

```
!
ipv6 snooping logging packet drop
!
ipv6 nd raguard policy HOST
!
interface Ethernet2/0
 description trunk to VMWare/kvm/etc.
 switchport
 switchport trunk encapsulation dot1q
 switchport mode trunk
 ipv6 nd raguard attach-policy HOST
```

```
!
```

IPv6 access port with device tracking and address logging

In this example we enable Device Tracking on port E0/0 and log the changes in the binding table to the syslog server - as well as logging the packets dropped by the FHS module.

```
!
ipv6 neighbor binding logging
ipv6 snooping logging packet drop
!
ipv6 snooping policy SNOOP
 tracking enable
!
!
interface Ethernet0/0
 description end host - no routers.
 switchport
 switchport access vlan 100
 switchport mode access
 ipv6 snooping attach-policy SNOOP
!
```

Untrusted host access port with source guard and very conservative filtering for legacy OSes

In this example we enable Source Guard on GigE1/0/1. We will only allow a port access that is found in the binding table. We have also protected the port from someone circumventing the filter with fragmented packets and undetermined transport.

```
!
ipv6 snooping policy SNOOP
 tracking enable
!
ipv6 source-guard policy SG
!
!
interface GigabitEthernet1/0/1
 description Host.
 switchport
 switchport access vlan 100
 switchport mode access
 ipv6 snooping attach-policy SNOOP
 ipv6 source-guard attach-policy SG
 ipv6 traffic-filter nofrags2 in
!
ipv6 access-list nofrags2
 ! The line below is too conservative for many modern OS environments !
 deny ipv6 any FE80::/64 fragments
 permit 1 any any
 permit 2 any any
 permit 3 any any
 . . . . repeat from 4 to 254 . . .
 permit 255 any any
!
```

"undetermined-transport" keyword support on various platforms

The access list above warrants some more explanation.

Some platforms may not support acl keyword "undetermined-transport". In that case they may either reject the command altogether, act erratically on such ACLs, or refuse to accept the ACL on the interface, like in the following example:

```
IPv6_FHS(config-if)#ipv6 traffic-filter nofrags in
% This ACL contains following unsupported entries.
% Remove those entries and try again.
    deny ipv6 any FE80::/64 undetermined-transport sequence 20
% This ACL can not be attached to the interface.
IPv6_FHS(config-if)#
```

In this case there is still a way to filter "undetermined transport" at the expense of a larger configuration. In this case we must simply apply the logic of "double negatives": instead of denying undetermined transport, we will permit all transports we can determine (the result will be the same!)

The modified configuration and access-list will look like this:

```
 !
 interface GigabitEthernet1/0/1
  ipv6 traffic-filter nofrags2 in
 !
 ipv6 access-list nofrags2
  !!!! Uncomment if using the legacy OS vulnerable to overlapping fragments
  ! deny ipv6 any FE80::/64 fragments
  permit 1 any any
  permit 2 any any
  permit 3 any any
  permit 4 any any
  permit 5 any any
  permit tcp any any
  permit 7 any any
  permit 8 any any
  permit 9 any any
  permit 10 any any
  permit 11 any any
  permit 12 any any
  permit 13 any any
  permit 14 any any
  permit 15 any any
  permit 16 any any
  permit udp any any
  permit 18 any any
  permit 19 any any
  permit 20 any any
  permit 21 any any
  permit 22 any any
  permit 23 any any
  permit 24 any any
  permit 25 any any
  permit 26 any any
  permit 27 any any
  permit 28 any any
  permit 29 any any
  permit 30 any any
```

```
permit 31 any any
permit 32 any any
permit 33 any any
permit 34 any any
permit 35 any any
permit 36 any any
permit 37 any any
permit 38 any any
permit 39 any any
permit 40 any any
permit 41 any any
permit 42 any any
permit 43 any any
permit 44 any any
permit 45 any any
permit 46 any any
permit 47 any any
permit 48 any any
permit 49 any any
permit esp any any
permit ahp any any
permit 52 any any
permit 53 any any
permit 54 any any
permit 55 any any
permit 56 any any
permit 57 any any
permit icmp any any
permit 59 any any
permit 60 any any
permit 61 any any
permit 62 any any
permit 63 any any
permit 64 any any
permit 65 any any
permit 66 any any
permit 67 any any
permit 68 any any
permit 69 any any
permit 70 any any
permit 71 any any
permit 72 any any
permit 73 any any
permit 74 any any
permit 75 any any
permit 76 any any
permit 77 any any
permit 78 any any
permit 79 any any
permit 80 any any
permit 81 any any
permit 82 any any
permit 83 any any
permit 84 any any
permit 85 any any
permit 86 any any
permit 87 any any
permit 88 any any
permit 89 any any
permit 90 any any
permit 91 any any
permit 92 any any
permit 93 any any
permit 94 any any
```

```
permit 95 any any
permit 96 any any
permit 97 any any
permit 98 any any
permit 99 any any
permit 100 any any
permit 101 any any
permit 102 any any
permit 103 any any
permit 104 any any
permit 105 any any
permit 106 any any
permit 107 any any
permit pcp any any
permit 109 any any
permit 110 any any
permit 111 any any
permit 112 any any
permit 113 any any
permit 114 any any
permit 115 any any
permit 116 any any
permit 117 any any
permit 118 any any
permit 119 any any
permit 120 any any
permit 121 any any
permit 122 any any
permit 123 any any
permit 124 any any
permit 125 any any
permit 126 any any
permit 127 any any
permit 128 any any
permit 129 any any
permit 130 any any
permit 131 any any
permit sctp any any
permit 133 any any
permit 134 any any
permit 135 any any
permit 136 any any
permit 137 any any
permit 138 any any
permit 139 any any
permit 140 any any
permit 141 any any
permit 142 any any
permit 143 any any
permit 144 any any
permit 145 any any
permit 146 any any
permit 147 any any
permit 148 any any
permit 149 any any
permit 150 any any
permit 151 any any
permit 152 any any
permit 153 any any
permit 154 any any
permit 155 any any
permit 156 any any
permit 157 any any
permit 158 any any
```

```
permit 159 any any
permit 160 any any
permit 161 any any
permit 162 any any
permit 163 any any
permit 164 any any
permit 165 any any
permit 166 any any
permit 167 any any
permit 168 any any
permit 169 any any
permit 170 any any
permit 171 any any
permit 172 any any
permit 173 any any
permit 174 any any
permit 175 any any
permit 176 any any
permit 177 any any
permit 178 any any
permit 179 any any
permit 180 any any
permit 181 any any
permit 182 any any
permit 183 any any
permit 184 any any
permit 185 any any
permit 186 any any
permit 187 any any
permit 188 any any
permit 189 any any
permit 190 any any
permit 191 any any
permit 192 any any
permit 193 any any
permit 194 any any
permit 195 any any
permit 196 any any
permit 197 any any
permit 198 any any
permit 199 any any
permit 200 any any
permit 201 any any
permit 202 any any
permit 203 any any
permit 204 any any
permit 205 any any
permit 206 any any
permit 207 any any
permit 208 any any
permit 209 any any
permit 210 any any
permit 211 any any
permit 212 any any
permit 213 any any
permit 214 any any
permit 233 any any
permit 234 any any
permit 235 any any
permit 236 any any
permit 237 any any
permit 238 any any
permit 239 any any
permit 240 any any
```

```
 permit 241 any any
 permit 242 any any
 permit 243 any any
 permit 216 any any
 permit 217 any any
 permit 218 any any
 permit 219 any any
 permit 220 any any
 permit 221 any any
 permit 222 any any
 permit 223 any any
 permit 224 any any
 permit 225 any any
 permit 226 any any
 permit 227 any any
 permit 228 any any
 permit 229 any any
 permit 230 any any
 permit 231 any any
 permit 232 any any
 permit 244 any any
 permit 245 any any
 permit 246 any any
 permit 247 any any
 permit 248 any any
 permit 249 any any
 permit 250 any any
 permit 251 any any
 permit 252 any any
 permit 253 any any
 permit 254 any any
 permit 255 any any
!
```

Note that if you are running a Unix-based system such as MacOS X, you can generate the sequence of "permit" statements with the following command from the OS console:

```
$ for i in `seq 1 255`; do echo permit $i any any; done
```

Further Reading

http://www.cisco.com/en/US/docs/ios-xml/ios/ipv6_fhsec/configuration/xe-3s/ip6f-xe-3s-book.pdf

https://ciscolive365.com/connect/search.ww?searchPhrase=BRKSEC-3003

http://www.cisco.com/en/US/docs/ios-xml/ios/ipv6_fhsec/configuration/15-s/ip6-snooping.html#GUID-8ABF862B-5D7D

Feedback

This document was edited by Andrew Yourtchenko in collaboration with Steve Simlo, Jim Bailey, Tim Martin and several other people. Thanks a lot to Fernando Gont for a thorough review and useful suggestions. Feel free to send us the feedback - ayourtch at cisco. This is the first version of the document and we plan to evolve it in the future - your feedback is an important factor which will determine how it evolves. Please let us know! Thank you!