Troubleshooting Unexpected Route Leaking in ACI

Contents

Overview

ACI handles many traditionally complex routing and switching configurations through the deployment of simple policies. Among these functionalities is the ability to leak routes between vrfs in order to facilitate shared services. Traditionally, this involved many steps such as defining route targets, creating BGP address families, route distinguishers, and replicating this configuration across many devices.

Within ACI route leaking is handled via the combination of contracts and setting specific shared flags on subnets. All of the traditional configuration that is required to make route-leaking work is handled on the backend as a result of of the contract and shared subnet configuration.

However, with this configuration abstracted, it can become more challenging to identify which contract is actually causing a route to be leaked. This is especially true in environments with large numbers of epg's, vrfs, and contracts. If a route is being unexpectedly leaked between vrfs how can an administrator identify which configuration (contract) is causing this to happen?

The purpose of this document is to demonstrate how to identify which contract relationship is causing a route in ACI to be leaked between VRFs. Its helpful to already be familiar with traditional route-leaking concepts such as route-targets and BGP VPNv4.

Software Used

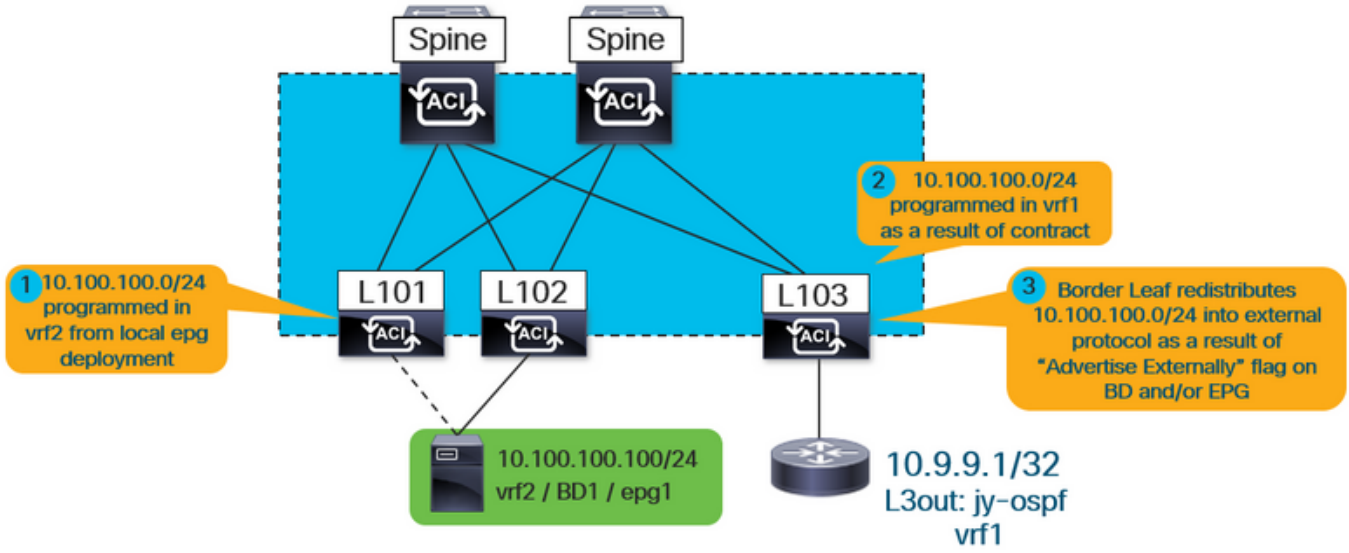
All examples in this document are based on aci software 4.2(3j).

Why is a Bridge Domain / EPG subnet from VRF x installed in VRF y?

This section will focus on the scenario where a BD or EPG subnet is being unexpectedly leaked to another vrf. For a BD / EPG subnet to be leaked the "Shared Between VRFs" flag must be configured. The more challenging part is understanding which contract is causing this to be leaked so that is what this section will address.

At a high-level this is the workflow for what happens when a BD / EPG subnet is leaked between vrfs.

Figure 1.



*Note that #3 is only applicable when advertising out a shared l3out. #1 and #2 always apply regardless of if a shared l3out is used or the shared services are entirely internal.

First of all, how can the user know if the installed route is leaked as a result of a BD or EPG subnet?

When running the "**show ip route vrf <name>**" the "**pervasive**" flag indicates that the route is a BD or EPG subnet.

For example, in the above topology this would be seen on the border leaf in the external vrf (vrf1):

```
leaf103# show ip route 10.100.100.100 vrf jy:vrf1
IP Route Table for VRF "jy:vrf1"
'*' denotes best ucast next-hop
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

10.100.100.0/24, ubest/mbest: 1/0, attached, direct, pervasive
    *via 10.3.144.68%overlay-1, [1/0], 21:29:54, static, tag 4294967292
         recursive next hop: 10.3.144.68/32%overlay-1
```

Additionally the destination vrf that the subnet was leaked from can be viewed by running the following command:

```
leaf103# vsh -c "show ip route 10.100.100.100 detail vrf jy:vrf1"
IP Route Table for VRF "jy:vrf1"
'*' denotes best ucast next-hop
```

```
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

10.100.100.0/24, ubest/mbest: 1/0, attached, direct, pervasive
    *via 10.3.144.68%overlay-1, [1/0], 21:34:16, static, tag 4294967292
         recursive next hop: 10.3.144.68/32%overlay-1
         vrf crossing information:  VNID:0x258003 ClassId:0x18 Flush#:0x2
```

*(note the vrf crossing information is set regardless of if the destination vrf is different than the lookup vrf).

In the above output the vrf crossing vnid is set to 0x258003 , or decimal 2457603. How can the vrf that vnid 2457603 belongs to be identified?

From the APIC simply query the fvCtx object and filter based on the segid.

```
apic1# moquery -c fvCtx -f 'fv.Ctx.seg=="2457603"'
Total Objects shown: 1

# fv.Ctx
name                 : vrf2
dn                   : uni/tn-jy/ctx-vrf2
pcEnfDir             : ingress
pcEnfPref            : enforced
pcTag                : 49153
scope                : 2457603
seg                  : 2457603
```

As expected, the route is being learned from the vrf2 vrf.

Its still unknown at this point which contract is being used as well as which epg is providing and which epg is consuming to cause this route to get installed. There are a couple of considerations to remember in regards to the provider and consumer relationship:
1. For an inter-vrf contract relationship the contract (and resulting zoning-rule) is only installed in the vrf of the consumer epg. As a result "show zoning-rule" in the provider vrf will not show the relationship.
2. Even though the contract is only installed in the consumer vrf, the provider vrf must get the route for the consumer vrf BD subnet, which means that the leaf must have some configuration reference to the contract.

Identify the contract when the route is being unexpectedly leaked to the consumer VRF

The ipCons object on the leaf is installed on the leaf which references...

a.) the route being leaked to the consumer vrf

b.) the contract establishing the relationship

c.) the provider and consumer epg's in the relationship.

In the below output "jy:vrf1" is the consumer vrf that the route is being leaked to and "10.100.100.0/24" is the route that is being leaked.

```
leaf103# moquery -c ipCons -f 'ip.Cons.dn*"jy:vrf1/rt-\[10.100.100.0/24\]"'
Total Objects shown: 1

# ip.Cons
consDn       : cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-
jy/brc-shared/dirass/cons-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-
shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]
subConsDn    :
childAction  :
dn           : sys/ipv4/inst/dom-jy:vrf1/rt-[10.100.100.0/24]/rsrouteToRouteDef-[bd-[uni/tn-
jy/BD-bd1]-isSvc-no/epgDn-[uni/tn-jy/ap-ap1/epg-epg1]/rt-[10.100.100.1/24]]/cons-[cdef-[uni/tn-
jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-jy/brc-shared/dirass/cons-
[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-
ap1/epg-epg1]-any-no]]-sub-[]
lcOwn        : local
modTs        : 2019-12-23T12:50:51.440-05:00
name         :
nameAlias    :
rn           : cons-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-
[uni/tn-jy/brc-shared/dirass/cons-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-
shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]]-sub-[]
status       :
```

From the above output the contract name is "shared", the consumer epg is l3out epg "uni/tn-jy/out-jy-ospf/instP-all" and the provider epg is "uni/tn-jy/ap-ap1/epg-epg1".

Identify the contract when the route is being unexpectedly leaked to the provider VRF

The consNode object is installed on the leaf in the provider vrf. It references the BD subnet in the consumer vrf that is being leaked, the contract, and the epg's within the relationship. Before querying this object find the BD subnet where the route is configured. This can be done by querying the fvSubnet object on the apic:

```
apic1:~> moquery -c fvSubnet -f 'fv.Subnet.dn*"10.100.100"'

# fv.Subnet
ip           : 10.100.100.1/24
dn           : uni/tn-jy/BD-bd1/subnet-[10.100.100.1/24]
preferred    : no
rn           : subnet-[10.100.100.1/24]
scope        : public,shared
```

The route is configured in the tn-jy/BD-bd1 bridge domain. Use this and the vnid of the provider vrf (that the route is being leaked to) to run the below command.

```
leaf103# moquery -c consNode -f 'cons.Node.dn*"2949122"' -f 'cons.Node.dn*"tn-jy/BD-bd1"'
Total Objects shown: 1

# cons.Node
consDn       : cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-
jy/brc-shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-
shared/dirass/cons-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]
annotation   :
childAction  :
descr        :
dn           : consroot-[bd-[uni/tn-jy/BD-bd1]-isSvc-no]-[sys/ctx-[vxlan-2949122]]/consnode-
[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-jy/brc-
shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-
```

```
shared/dirass/cons-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]]
extMngdBy    :
lcOwn        : local
modTs        : 2019-12-23T12:25:36.153-05:00
name         :
nameAlias    :
rn           : consnode-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-
all]/fr-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-
jy/brc-shared/dirass/cons-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]]
status       :
uid          : 0
```

From the above output the contract name is "shared", the consumer epg is "uni/tn-jy/ap-ap1/epg-epg1" and the provider epg is l3out "tn-jy/out-jy-ospf/instP-all".

Identify the contract when the route is being unexpectedly leaked by a consumed vzAny contract

The vzAny example is going to be identical from a verification perspective to a traditional provider / consumer relationship. The below examples will just demonstrate what this would look like. Note that an inter-vrf contract is only supported with the vzAny as the consumer.

vzAny Example 1: Route unexpectedly leaked to consumer VRF

Similar to the first example looked at where verification was done in the consumer vrf, the ipCons object will be used again.

```
leaf103# moquery -c ipCons -f 'ip.Cons.dn*"jy:vrf1/rt-\[10.100.100.0/24\]"'
Total Objects shown: 1

# ip.Cons
consDn       : cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/ctx-vrf1/any]/fr-[uni/tn-jy/brc-
shared/any-[uni/tn-jy/ctx-vrf1/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf1/any]-any-yes]/to-
[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]
subConsDn    :
childAction  :
dn           : sys/ipv4/inst/dom-jy:vrf1/rt-[10.100.100.0/24]/rsrouteToRouteDef-[bd-[uni/tn-
jy/BD-bd1]-isSvc-no/epgDn-[uni/tn-jy/ap-ap1/epg-epg1]/rt-[10.100.100.1/24]]/cons-[cdef-[uni/tn-
jy/brc-shared]/epgCont-[uni/tn-jy/ctx-vrf1/any]/fr-[uni/tn-jy/brc-shared/any-[uni/tn-jy/ctx-
vrf1/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf1/any]-any-yes]/to-[uni/tn-jy/brc-
shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]]-sub-[]
lcOwn         : local
modTs         : 2019-12-23T13:11:08.077-05:00
name          :
nameAlias     :
rn            : cons-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/ctx-vrf1/any]/fr-[uni/tn-
jy/brc-shared/any-[uni/tn-jy/ctx-vrf1/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf1/any]-any-
yes]/to-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]]-sub-[]
status        :
```

From the above output the contract name is "shared", the consumer epg is the vrf1 vzAny "tn-jy/ctx-vrf1/any" and the provider epg is "uni/tn-jy/ap-ap1/epg-epg1".

vzAny Example 2: Route unexpectedly leaked to provider VRF

Similar to the second example looked at where verification was done in the provider vrf, the consNode object will be used again. Remember to get the bd name of the BD where the leaked

subnet is configured and the vnid of the vrf it is leaked to.

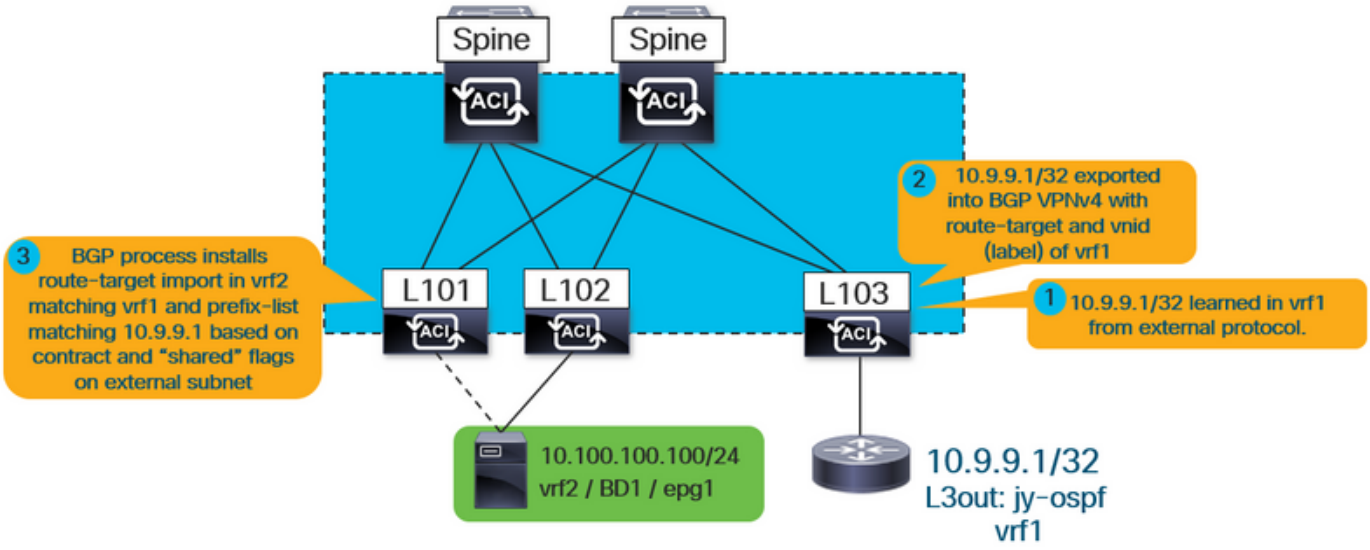
```
leaf103# moquery -c consNode -f 'cons.Node.dn*"vxlan-2949122"' -f 'cons.Node.dn*"tn-jy/BD-bd1"'
Total Objects shown: 1

# cons.Node
consDn       : cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-
jy/brc-shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-
shared/any-[uni/tn-jy/ctx-vrf2/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf2/any]-any-yes]
annotation   :
childAction  :
descr        :
dn           : consroot-[bd-[uni/tn-jy/BD-bd1]-isSvc-no]-[sys/ctx-[vxlan-2949122]]/consnode-
[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-all]/fr-[uni/tn-jy/brc-
shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-jy/brc-shared/any-
[uni/tn-jy/ctx-vrf2/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf2/any]-any-yes]]
extMngdBy    :
lcOwn        : local
modTs        : 2019-12-23T13:06:09.016-05:00
name         :
nameAlias    :
rn           : consnode-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/out-jy-ospf/instP-
all]/fr-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]/to-[uni/tn-
jy/brc-shared/any-[uni/tn-jy/ctx-vrf2/any]-type-cons_as_any/cons-[uni/tn-jy/ctx-vrf2/any]-any-
yes]]
status       :
uid          : 0
```

From the above output the contract name is "shared", the consumer epg is the vrf2 vzAny "tn-jy/ctx-vrf2/any" and the provider epg is l3out "tn-jy/out-jy-ospf/instP-all".

Why is an external route from VRF y installed in VRF x?

At a high-level this is the workflow for what happens when an l3out-learned (external) route is leaked between vrfs.

Figure 2.



As can be seen above, the internal vrf (vrf2 in the this case) installs a route-target import that matches vrf1. It also installs an import map on the bgp process that should have prefix-list entries

matching everything defined in the l3out that has the "shared route control subnet" flag selected.

Regardless of which epg is the provider or consumer, the verification steps are the same because the contract will always be responsible for causing the route-target import and the corresponding prefix-lists which will leak the routes to get installed.

First of all, validate that the route is in fact being learned through an l3out:

```
leaf101# show ip route 10.9.9.1 vrf jy:vrf2
IP Route Table for VRF "jy:vrf2"
'*' denotes best ucast next-hop
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

10.9.9.1/32, ubest/mbest: 1/0
    *via 10.3.248.4%overlay-1, [200/5], 00:00:13, bgp-65001, internal, tag 65001
```

In the above example, the fact that it is learned from the fabric bgp process pointing to another leaf in the overlay indicates this came from an l3out.

Run the following information to get more info about which vrf it was learned from:

```
leaf101# vsh -c "show ip route 10.9.9.1 detail vrf jy:vrf2"
IP Route Table for VRF "jy:vrf2"
'*' denotes best ucast next-hop
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

10.9.9.1/32, ubest/mbest: 1/0
    *via 10.3.248.4%overlay-1, [200/5], 00:05:46, bgp-65001, internal, tag 65001 (mpls-vpn)
         MPLS[0]: Label=0 E=0 TTL=0 S=0 (VPN)
         client-specific data: 6b
         recursive next hop: 10.3.248.4/32%overlay-1
         extended route information: BGP origin AS 65001 BGP peer AS 65001 rw-vnid: 0x2d0002
table-id: 0x17 rw-mac: 0
```

As shown ealier in this document, rewrite vnid 0x2d0002 / 2949122 is the destination vrf. The rw-vnid value being set to a non-zero value in an external route example indicates that this was learned from a different vrf. Running **moquery -c fvCtx -f 'fv.Ctx.seg=="2949122"'** on the apic would indicate that this belongs to vrf1.

Next, find the route-target imports as well as the import route-map that is tied to the bgp process.

```
leaf101# show bgp process vrf jy:vrf2

Information regarding configured VRFs:

BGP Information for VRF jy:vrf2
VRF Type                        : System
VRF Id                          : 23
VRF state                       : UP
VRF configured                  : yes
VRF refcount                    : 0
VRF VNID                        : 2457603
Router-ID                       : 10.100.100.1
Configured Router-ID            : 0.0.0.0
```

```
Confed-ID                       : 0
Cluster-ID                      : 0.0.0.0
MSITE Cluster-ID                : 0.0.0.0
No. of configured peers         : 0
No. of pending config peers     : 0
No. of established peers        : 0
VRF RD                          : 101:2457603
VRF EVPN RD                     : 101:2457603

    Information for address family IPv4 Unicast in VRF jy:vrf2
    Table Id                    : 17
    Table state                 : UP
    Table refcount              : 5
    Peers      Active-peers    Routes     Paths      Networks   Aggregates
    0          0               2          2          0          0

    Redistribution
        None

    Wait for IGP convergence is not configured
    Import route-map 2457603-shared-svc-leak <-- bgpRtCtrlMapP
    Export RT list:
        65001:2457603
    Import RT list:
        65001:2457603
        65001:2949122 <-- bgpRttEntry
    Label mode: per-prefix
```

The internal vrf above is exporting and importing its own route-target (65001:2457603). It is also importing 65001:2949122. The 2949122 RT corresponds to the vrf vnid which it is importing (vrf1). bgpRtCtrlMapP is the object name for the import route-map which contains the prefix-lists. bgpRttEntry is the object name for the import route-target.

Next, using the vnid of the internal vrf that is learning the external vrf routes, query all of the prefix-lists that are installed within the shared services route-map.

```
leaf101# moquery -c rtpfxEntry -f 'rtpfx.Entry.dn*"pfxlist-IPv4'.*'2457603-shared-svc-leak"' |
egrep "criteria|dn|pfx|toPfxLen"
# rtpfx.Entry
criteria         : inexact
dn               : sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak/ent-2
pfx              : 0.0.0.0/0
toPfxLen         : 32
# rtpfx.Entry
criteria         : exact
dn               : sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak/ent-3
pfx              : 10.9.9.1/32
toPfxLen         : 0
# rtpfx.Entry
criteria         : exact
dn               : sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak/ent-1
pfx              : 10.9.9.0/24
toPfxLen         : 0
```

Each entry should correspond to an external subnet. The "exact / inexact" attribute indicates whether the "aggregate shared" flag was set on the external subnet. The 0.0.0.0/0 prefix with the inexact flag indicates that it would match all routes that are more specific (effectively everything). The 10.9.9.0/24 prefix with the exact flag indicates it would only match that /24.

Find the entry (or entries) that matches the route that is being unexpectedly leaked. In this case the prefix is 10.9.9.1/32 would be matched by ent-2 and ent-3 in the above outputs.

Using the prefix-list name, find the sequence number within the route-map that matches it.

```
leaf101# moquery -c rtmapRsRtDstAtt -f 'rtmap.RsRtDstAtt.tDn*"pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak"'
Total Objects shown: 1

# rtmap.RsRtDstAtt
tDn          : sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak
childAction  :
dn           : sys/rpm/rtmap-2457603-shared-svc-leak/ent-1001/mrtdst/rsrtDstAtt-[sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak]
forceResolve : yes
lcOwn        : local
modTs        : 2019-12-24T11:17:08.668-05:00
rType        : mo
rn           : rsrtDstAtt-[sys/rpm/pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak]
state        : formed
stateQual    : none
status       :
tCl          : rtpfxRule
tSKey        : IPv4-2949122-24-25-2457603-shared-svc-leak
tType        : mo
```

The above output shows that this is route-map entry 1001. The final portion here is to understand which contract was responsible for creating route-map entry 1001 within the 2457603-shared-svc-leak route-map. This can be queried on the leaf from the fvAppEpGCons object.

```
leaf101# moquery -c fvAppEpGCons -f 'fv.AppEpGCons.dn*"rtmap-2457603-shared-svc-leak/ent-1001"'
Total Objects shown: 1

# fv.AppEpGCons
consDn       : cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/ap-ap1/epg-epg1]/fr-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]/to-[uni/tn-jy/brc-shared/dirass/cons-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]
childAction  :
descr        :
dn           : uni/ctxrefcont/ctxref-[sys/ctx-[vxlan-2457603]]/epgref-[uni/tn-jy/ap-ap1/epg-epg1]/epgpol-[sys/rpm/rtmap-2457603-shared-svc-leak/ent-1001]/epgcons-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/ap-ap1/epg-epg1]/fr-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]/to-[uni/tn-jy/brc-shared/dirass/cons-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]]
lcOwn        : local
modTs        : 2019-12-23T14:36:48.753-05:00
name         :
nameAlias    :
ownerKey     :
ownerTag     :
rn           : epgcons-[cdef-[uni/tn-jy/brc-shared]/epgCont-[uni/tn-jy/ap-ap1/epg-epg1]/fr-[uni/tn-jy/brc-shared/dirass/prov-[uni/tn-jy/ap-ap1/epg-epg1]-any-no]/to-[uni/tn-jy/brc-shared/dirass/cons-[uni/tn-jy/out-jy-ospf/instP-all]-any-no]]
status       :
```

The above output shows that the contract name is "shared", the provider epg is "tn-jy/ap-ap1/epg-epg1" and the consumer l3out epg is "tn-jy/out-jy-ospf/instP-all"

Summary

Route Leaked from BD / EPG Subnet

If a leaked route has the "pervasive" flag set in "show ip route" then it is leaked from a BD/EPG subnet configured. The following two commands can be used to check which contract relationship is causing this to be leaked. They would be run on the leaf where the route is unexpectedly installed.

If the vrf where the route is unexpectedly leaked is the consumer:
moquery -c ipCons -f 'ip.Cons.dn*"jy:vrf1/rt-\[10.100.100.0/24\]"' <--*jy:vrf1 is the name of the vrf the route is leaked to, the route is 10.100.100.0/24*

If the vrf where the route is unexpectedly leaked is the provider:
moquery -c consNode -f 'cons.Node.dn*"2949122"' -f 'cons.Node.dn*"tn-jy/BD-bd1"' <--*2949122 is the vnid of the vrf the route is leaked to, tn-jy/BD-bd1 is the name of the BD where the subnet is configured (within the vrf the route is leaked from).*

Route Leaked from L3out

If the leaked route is learned through the internal fabric iBGP process and running **vsh -c "show ip route x.x.x.x/y detail vrf <name>"** shows a non-zero rw-vnid value then the route is being learned from an l3out in another vrf. Validation is the same regardless of which epg is the consumer and which is the provider.

1. *Identify the shared services import route-map on the internal vrf bgp process:*
show bgp process vrf jy:vrf2 | grep "Import route-map" <--*jy:vrf2 is the internal vrf that the route is leaked to*

2. Identify the prefix-list that is within the shared services route-map that matches the leaked route:
moquery -c rtpfxEntry -f 'rtpfx.Entry.dn*"pfxlist-IPv4'.*'2457603-shared-svc-leak"' | egrep "criteria|dn|pfx|toPfxLen" <--*2457603 is the vnid of the internal vrf in this example*

3. After finding which prefix list(s) reference the route, identify which route-map sequence number references the list(s):
moquery -c rtmapRsRtDstAtt -f 'rtmap.RsRtDstAtt.tDn*"pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak"' <--*pfxlist-IPv4-2949122-24-25-2457603-shared-svc-leak is the prefix-list name*

4. Using the rtmap and entry number run the following command to find out which contract relationship pushed that route-map entry:
moquery -c fvAppEpGCons -f 'fv.AppEpGCons.dn*"rtmap-2457603-shared-svc-leak/ent-1001"' <--*rtmap-2457603-shared-svc-leak/ent-1001 is the route-map name and entry number from step 3.*