Stack Instances Interface Document
PSARC 2006/366
07/19/06
Erik Nordmark

1 Introduction

Stack instances introduces the ability to have separate zones connected to separate LANs or VLANs without introducing any network security issues. It does this by applying a simple approach to separate all the previously global variables in the TCP/IP collection of kernel modules, so that there can be multiple instances of them, and by providing some extensions to zonecfg to specify that a zone should have its own TCP/IP stack. In its simplest form what is needed in zonecfg is to add

```
set stacktype=exclusive
```

This addition to zonecfg is necessary because for compatibility reasons we do not believe we can change the existing behavior in zones networking, especially not in a Solaris 10 Update. Thus we can have a combination of “shared-stack zones” (today's S10 behavior) and the new “exclusive-stack zones” that are configured by setting the above zonecfg property.

An implication of having a separate TCP/IP stack per exclusive-stack zone is that all aspects of the TCP/IP state and policies become per stack. This includes the IP routing table. ARP table, IPsec policies and associations, IP Filter rules, TCP/IP ndd variables.

2 Background – Networking and Zones

As customers are starting to deploy using Zones, some of them see a good fit was the design center for the Zones networking support; the servers to be consolidated are on the same IP subnet.

Other customers see some problems or limitations.

The problems stem from desiring to consolidate applications that need to communicate on different subnets that are on different VLANs or different LANs. The customer's we've talked that express the need for

- separate routing per zone
- prevent loopback traffic between zones inside the server

all have the expectation (and in some cases the explicit requirement) that the network traffic for a zone must be completely separate from the traffic for other zones; they might share the same physical NIC using VLANs, but that's the only allowed degree of sharing of the network.

Figure 1 shows a conceptual model of how Zones networking works in S10, with conceptually a separate TCP, UDP and SCTP for each zone, and incoming packets being demultiplexed to the separate transports based on their destination IP address. For outgoing packets, they end up with a source IP address which is one of the IP addresses assigned to the zone.

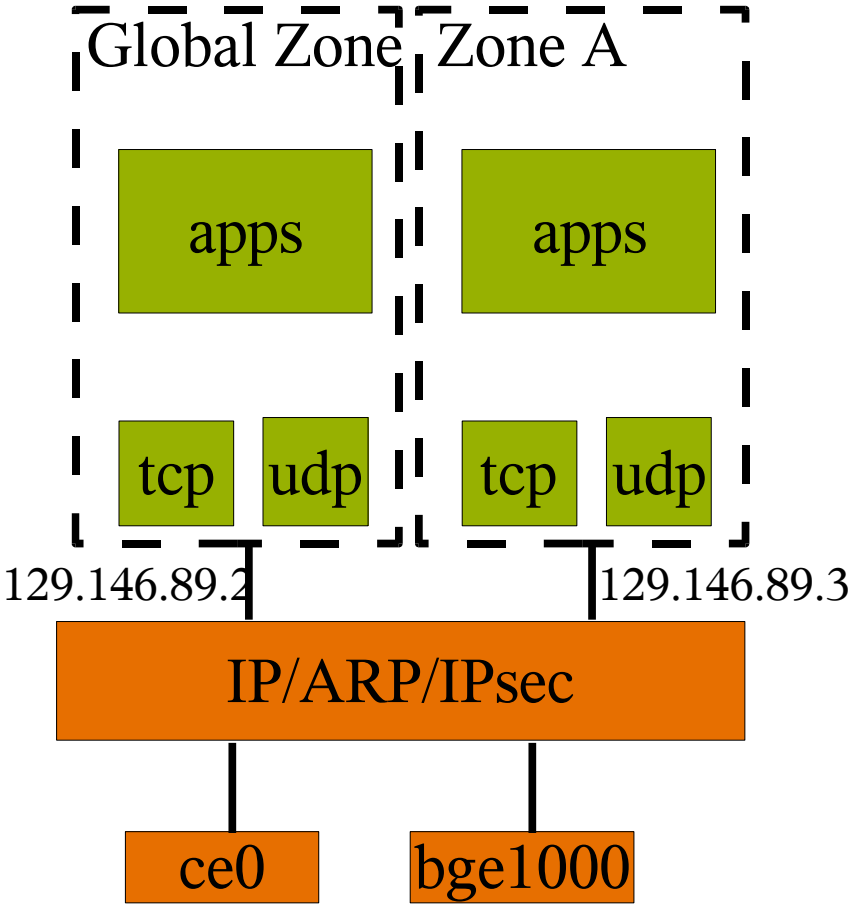

Figure 1: Conceptual model of zones networking

What zones provides for networking is name space isolation for the TCP/UDP/SCTP port name space by giving each zone its own distinct IP address(es). When the server is connected to multiple network interfaces, the shared IP behaves just as it behaved prior to the introduction of Zones; packets are routed as best viewed by IP. But what is needed when consolidating applications that need to be connected to isolated (V)LANs is in fact isolation for network connectivity. In the case of Solaris such isolation can be expressed in the form of which network interface names are accessible in each zone (since VLANs and other layer 2 concepts appear as separate network interface names). Then for each such zone there needs to be no internal IP-level sharing as depicted in figure 2.

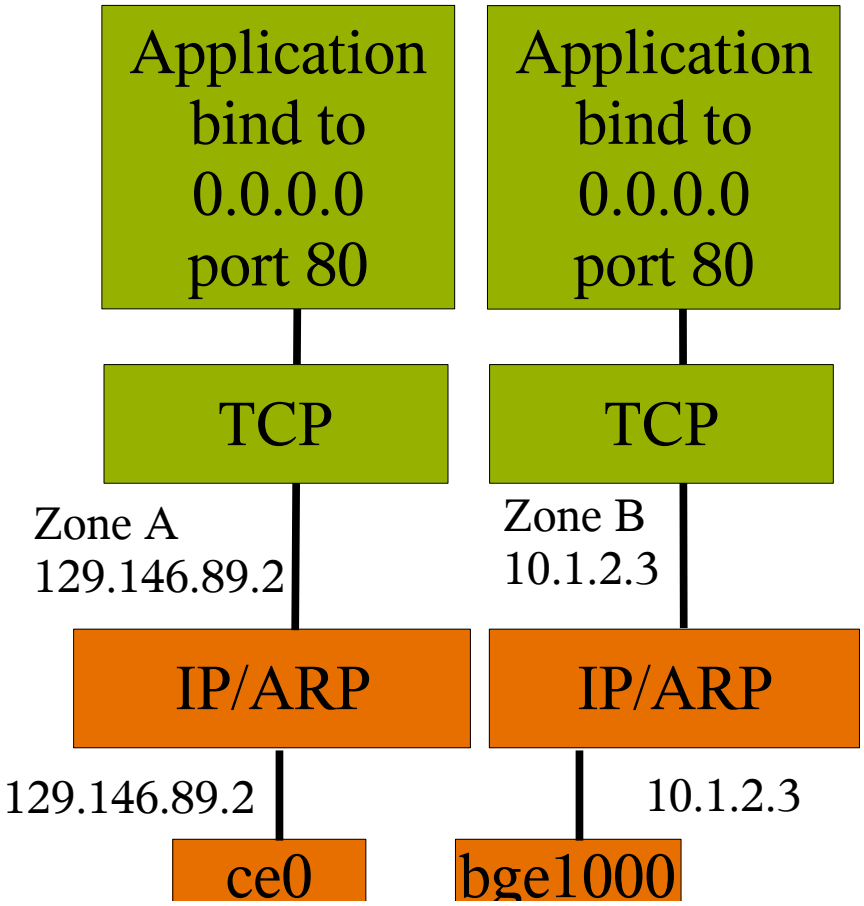

Figure 2: Conceptual model of zones networking with IP-level isolation

In many cases customers try to make their cases of different subnets and different (V)LANs work through some manual tweaking such as manually added routes and a global zone with IP address 0.0.0.0 on some NICs. But the Zones networking support was designed with a shared IP stack in mind and just separate IP address(es) per zone. Thus while Solaris ensures that a zone can only send IP packets with its IP addresses, and can only receive unicast packets address those IP addresses, there are separation issues when looking from the network in towards Solaris. Their implications are that:

- Even though some S10 RFEs were implemented to try to allow separate default routes per zone, this is hard to configure and doesn't work as expected. In some configurations it isn't possible to get working routing when separate default routes per zone are used.
- We can not guarantee that packets can not accidentally show up in a different zone that some security policy requires; there are just too many protocol features and code paths in IP, ARP, TCP, etc that can cause such "leakage" for packets arriving from the network.
- If/when the customer assumes that a zone can only send packets on the NICs on which they have IP addresses they become surprised. The design was to restrict the source addresses that a zone can use when sending packets, and the relationship between the NIC on which that IP address is configured and where packets get sent is quite weak and absent in some cases. (E.g., sending multicast or broadcast packets, sending ICMP errors and TCP resets, and sending packets after they have been queued waiting for ARP.)

- The behavior after the manual tweaking isn't guaranteed to work; we have no way to ensure that some customer invented approach or some blog suggestion will continue to work after a patch.
- The behavior with the manual tweaking is very fragile due to the “one bugfix at a time” evolution of the zones networking during S10 development. Today we are discovering unintended consequences of some of the code changes that were made so that different zones could use different default routes.

Note that some of the customers might express the issues they see as "it is too difficult to configure" a zone to have different default routes and get it to work, but the more fundamental issue seems to be that the expect secure separation (which is why they are using separate LANs or VLANs on the network), but they don't seem to see the security holes implicit in sharing the same IP and ARP for all the zones.

The result of this is that we are already starting to see escalations including escalated RFEs and these escalations can be extremely difficult to solve.

Part of the root cause for those is that Sun has and still do express networking aspects of zones as “separate networking” even though the separation is limited to the port number space. But with such vague descriptions combined with Zones overall being described as providing isolation, it isn't surprising if customers and the field assume that when zones are combined with network isolation (using separate LANs or VLANs), that the result will be isolation end-to-end from the network to the applications. In fact it takes a while to explain that even though the (V)LANs are separate and the applications are in separate zones, there is a shared IP in Solaris in between those whose role is to connect things together.

3 Justification

It is clear that a significant part of the customers that do application consolidation in the datacenter have a need to provide application isolation in combination with separate (V)LANs. Due to our unclear communication and the business needs, in some cases we end up having to address escalated RFEs. Thus one aspect of the business justification is that we must stop the bleeding and avoid wandering further into a support problem by providing an alternative to the S10 "shared stack" model. That translates into a desire to make this project available in the earliest possible S10 update.

We also have an opportunity to set us apart from the competition by leveraging the exclusive stacks to provide new capabilities such as:

- Makes networking with zones look more similar to Xen, which means that we can leverage work (e.g., all the GLDv3 work) across Zones and Xen
- Easier testing of Solaris networking; can run multiple exclusive stacks inside one box and run automated testing that today requires several systems that are manually cabled together
- The potential to build more secure infrastructure by having a zone's sole connection to the network be an IPsec tunnel that is provided by the global zone
- The potential to leverage more network security pieces such as packet filtering capability for Zones as well as for Xen virtualization, by placing filtering hooks in the G:LD framework.
- The potential to have others leverage Solaris to build virtual routers or other virtual network appliances where each appliance has an presence on the network which is independent of other appliances on the same box.

The BrandZ team also desires a supported way to run NAT in the global zone for a branded zone, so that only one IP address is needed for the system. Other approaches to providing NAT functionality in

such configurations do not appear to be supportable due to the complex interdependencies that one would have to create inside the single IP stack. Thus the shortest time to market for such supported functionality is having an exclusive stack for the BrandZ zone, and have that communicate via the NAT in the global zone using an local-to-the box pseudo Ethernet (called vSwitch; something that is part of the Crossbow precut).

Note that in order to apply this approach to BrandZ, the branded zone has to do its own IP layer configuration (i.e., issue ifconfig or DHCP operations) since the global zone will not configure IP for a zone with an exclusive stack.

4 Goals

The goals of the project is to provide TCP/IP stack separation for non-global zones while preserving the ability to have zones shared the same TCP/IP stack as the global zone. The kernel modules that will be modified to support stack instances is conceptually just IP. But since IP is made up of several components (such as IPsec, routing sockets, IP Filter), and other code such as the transport protocols do direct function calls into the IP module, the set of kernel modules expands to:

- ip, rts, arp, ipsec, ipsecesp, ipsecah, keysock, spdsock
- tcp, udp, sctp, icmp
- ipf, pfil

Specifying whether a shared or an exclusive stack is used will be done in zonecfg, to preserve the current approach to configuring networking for zones.

Since the DHCP client code and IPv6 stateless address autoconfiguration works unmodified in a zone with an exclusive stack, zonecfg is also modified to be able to specify that a dynamic IP address configuration be used.

The following particular items will work per exclusive stack zone (without any specific changes – just from virtualizing the TCP/IP kernel state):

- DHCPv4 and IPv6 stateless address autoconfiguration
- IP Filter including its NAT functionality
- IPMP and CGTP
- IP routing
- ndd for setting TCP/UDP/SCTP as well as IP/ARP-level knobs
- IPsec

The security of the resulting system should be at least as good as with zones in S10. This includes limiting the ability of users in non-global zones causing harm to others on the same machine, and causing harm to other hosts on the network, and also the ability for network attackers to cause harm to the OS and the applications running on it.

5 Non-goals

No virtualization of the IP Policy Framework and IPQoS. Those can still be used in the global zone to manage shared-stack zone's IP traffic, but their low usage and possible replacement by Crossbow network resource controls means there is no business justification for virtualization them.

No GLD or other layer 2 changes. This means that for instance it will not be possible for an exclusive

stack zone that has access to bge1001 and bge1002 to create an aggregation of those two; layer 2 aggregations must be created in the global zone. The project works with device drivers (current and future) that provide a /dev name for the interface e.g., /dev/bge0 today or /dev/net/bge3001 in the future (future names being provided by the Clearview project).

The project will not address any kernel components above the transport layer (such as NFS, KSSL, NL7C). Such components, when and if they are made zones aware, do not and should not be aware of stack instances; whether the IP addresses assigned to a zone is from an exclusive or shared stack is immaterial to components above the transport layer.

No change to how IP is configured in Solaris, in particular, this project does not take the various networking /etc files (/etc/hostname.<ifname>, /etc/defaultrouter, /etc/resolv.conf) and replace them by something else like SMF properties. A result of this is that for exclusive stack zones we rely on the /etc files to configure the exclusive stack.

The stack instance concept only need to apply to kernel code that is "near" IP, because IP is the thing that is shared in the Solaris 10 zones model for networking. For instance, network applications are automatically separated for zones since each process runs in a zone. Furthermore, if kernel networking pieces that sit above UDP, TCP, SCTP, or RAWIP need to be virtualized for zones, they do not need to be aware of stack instances; such software can be virtualized by using the zoneid as the discriminator which applies whether or not IP is virtualized. Kernel software that is below IP, such a GLD or other network device drivers, doesn't have a need to be aware of stack instances either. Zones uses the /dev/ entry points as the way to control what each zone can access, and the device drivers already keep different instances (bge0 vs. bge1) apart. The only implication of this assumption is that is it sufficient to make the internal support routines (netstack framework) be project private.

We assume that there is no business need to make 3rd party software that sit between IP and the device drivers aware of zones and/or stack instances. This means that things like Firewall-1 and the Cisco VPN client can only be used in the shared stack (used by the global zone.)

6 Technical components

There are six primary technical components:

1. Introducing a model where a non-global zone can either share the IP stack with the global zone (which is the only possibility in S10), or have an exclusive IP stack to itself. A zone with an exclusive IP stack will need exclusive access to one or more network interfaces (could be separate LANs like bge1, or separate VLANs like bge2000; any datalink interface with a separate name in /dev/ can be used.)

   In this model, by design there is no sharing e.g., a zone with an exclusive stack has its own routing table, arp table, IPsec security policy and security associations, IP Filter rules and state, etc.

2. Extensions to zonecfg syntax to
   - Allow specifying that a zone has an exclusive IP stack: 'set stacktype=exclusive'
   - Since with exclusive stacks we also get the ability to configure a non-global zone with DHCP and/or IPv6 stateless address autoconfiguration, there are zonecfg syntax additions for this.
   - Allow specifying static default routers for zones with exclusive stacks.
   - A new 'restrict' boolean property for the 'net' resource which is passed to the kernel in form

of an exact match of the network interface name and the IP address. When set, this prevents a non-global zone that has an exclusive stack from "stealing" somebody else's IP address.

3. Splitting the PRIV_SYS_NET_CONFIG privilege by introducing the new PRIV_SYS_IP_CONFIG privilege, that allows a subset of the operations allowed by PRIV_SYS_NET_CONFIG.

   A zone with a global stack will be granted PRIV_SYS_IP_CONFIG and PRIV_SYS_NET_RAWACCES.

4. Changes to the networking SMF method scripts which today skip sections of the script if "zonename != global", to not skip those sections if the non-global zone has an exclusive stack. This test requires a way for a script to be able to ask "does this zone have an exclusive stack". The current approach is do to this with a new option for zonename(1m) - zonename -t.

5. Modifications to all the kernel modules that make up "IP" (such as /kernel/drv/ip and the IPsec modules), as well as all the kernel modules that perform function calls into IP (such as TCP and IP Filter), to allow multiple instances of all their writable global data.

6. A "netstack" kernel framework (netstack.h/netstack.c) which isolates the TCP/IP kernel modules from the mapping between zones and stacks, and provides things like kstat_create_netstack() analogous to kstat_create_zone(). For further details on this, see below.

7 Relationship between stacks and zones

As depicted in figure 2, there can both be zones that use the shared stack, and there can be zones which have their own exclusive stack. Those two approaches are sufficient for our current needs. In the future, it might be possible to remove the use of the shared stack but this requires more carefully understanding whatever dependencies the administrator has on the current zones networking behavior.

The project has carefully tried to isolate the knowledge of the relationship between zones and stacks so that, should there be need to solve different problems in this space in the future, we can easily build different relationships with minimal code change.

8 Configuration

The configuration aspects consists of extensions to zonecfg syntax to not only configure zones with exclusive stacks, but also to be able to specify static default routers, specify use of DHCP and/or IPv6 stateless address autoconfiguration, and not be able to restrict the IP addresses an exclusive stack zone can use.

9 zonecfg

The additions to zonecfg syntax are:

- New “stacktype” global property which can be set to “shared” (the default if not specified) or “exclusive”
- New “af” property for the “net” resource, which is used when an address property is not specified (dynamic address configuration) in order to select whether IPv4 or IPv6 should be configured.
- New “router” resource that allows specifying static default routers for zones with exclusive stacks.

- A new 'restrict' boolean property for the 'net' resource which prevents the exclusive stack zone from changing its IP address.

A result of these changes is that the output of zonecfg export changes.

The following examples shows how this changes things.

In S10s syntax we have something like

```
set zonepath=/export/zone1
add net
        set physical=bge1
        set address=10.1.2.3
```

If the global administrator wants zone1 be the only zone that uses bge1, then with stack instances the needed configuration will be:

```
set zonepath=/export/zone1
set stacktype=exclusive
add net
        set physical=bge1
        set address=10.1.2.3
add router
        set address=10.0.0.1
```

with the green italic text above shows the additional pieces. Note that in S10 if the administrator tried to have bge1 be exclusively used by zone1 without stack instances, the admin would have had to create some scripts on her own to ifconfig plumb bge0 0.0.0.0 in the global zone and also add the default route using that script.

If the administrator wants to prevent the non-global admin from changing the IP address or do 'arp -s pub', then she can add

```
set restrict=true
```

to the above net resource.

If the administrator instead wants to have the zone be configured using DHCP the configuration would be

```
set zonepath=/export/zone1
set stacktype=exclusive
add net
        set physical=bge1
        set af=inet
```

And finally, if the global admin wants to have the non-global zone use dynamic address configuration

for both IPv4 and IPv6, the configuration would be:

```
set zonepath=/export/zone1
set stacktype=exclusive
add net
        set physical=bge1
        set af=inet
add net
        set physical=bge1
        set af=inet6
```

10 zoneadm

There are no interface changes for zoneadm except additional error messages should an exclusive stack zone have a network physical which is also configured to be used by another zone that is booted.

11 zonename

We have added a -t option to zonename that makes it print the stacktype thus it outputs either “shared” or “exclusive”. This is used by the SMF method scripts below. (Perhaps there is a better way to do this than to overload zonename.)

12 SMF method scripts

The set of SMF services is unchanged, but for IP Filter the SMF methods appear in all zones as a result of the packaging changes.

Prior to stack instances a set of SMF method scripts (such as net-physical) had explicit checks for zonename == global, and in the case of a non-global zone the script would do very little if anything. Those 4 scripts are modified to instead check if the zone needs IP level configuration i.e. they check for zonename == global or zonename -t == shared.

The remainder of the scripts are unchanged that is they continue to use /etc/hostname.<ifname> etc.

When zonecfg/zoneadm installs and boots a zone, the zone's network configuration is used to create the /etc/ files for network configuration. Thus the configuration continues to be driven from zonecfg.

Ideally we'd like zonecfg to be able to control certain parameters (such as the IP addresses and default routers) but retain the flexibility of the local zone administrator controlling other aspects of IP configuration (just like the local zone administrator controls /etc/nsswitch.conf and /etc/resolv.conf). But due to /etc/hostname.<ifname> being overloaded as a possible repository for anything that can be set with ifconfig (e.g., MTU) this is cumbersome. We have a few options:

1. Have zoneadm create /etc/hostname.<ifname>, /etc/defaultrouter, /etc/dhc.<ifname> when the zone is installed but leave them untouched after this. This allows the non-global admin adding e.g. “mtu 1024” to the end of /etc/hostname.<ifname>, but zonecfg can't be used to change the IP address.
2. Have zoneadm create those files each time the zone is booted (using the technique invented in

Zulu of fork+zone_enter to avoid security issues). This means that any non-global admin changes to /etc/hostname.<ifname> are lost each time the zone is rebooted.

3. Have the /etc/hostname.<ifname> file be edited each time the zone is booted so that the current zonecfg settings are applied without removing any other configuration from the file. That would entail more or less re-implementing the ifconfig parser outside of ifconfig, which is unattractive.

Since the hope is that a future project (such as Network Automagic) will take the IP configuration out of the /etc/ files and into the SMF repository, it seems even more unattractive to take the 3rd approach.

13 Netstack framework (project private)

The netstack framework provides a way for the TCP/IP kernel modules to be aware of each stack instance that is created and destroyed as part of zones being created and destroyed. Thus it avoids the need for each TCP/IP module to track which zones have exclusive stacks and which use the shared stack. It also provides interfaces (kstat_create_netstack()) that takes care of making the kstats available in the correct zones, for both the shared stack case and the exclusive stack case.

The netstack framework is implemented using zone_key_create(), kstat_create_zone(), kstat_zone_add(), etc. Thus it sits as a veneer between the TCP/IP kernel modules and the zones code providing the abstraction of a netstack_t.

In order to provide efficient linkage between different parts of the TCP/IP code, for instance TCP needing to call ire_route_lookup(), and since we don't need this framework to be extensible outside of the set of kernel modules which perform function calls into the TCP/IP code, the netstack framework is less general that the Zone zsd support in that each kernel module that uses it requires a #define of a moduleid in netstack.h and a recompile of the kernel.

The netstack functions provided to the rest of the TCP/IP code are:

```
void netstack_register(int moduleid,
 void *(*module_create)(netstackid_t, netstack_t *),
 void (*module_shutdown)(netstackid_t, void *),
 void (*module_destroy)(netstackid_t, void *));
void netstack_unregister(int moduleid);
```

The above functions are called from a modules _init() and _fini() functions.

The following functions are used e.g., by a TCP/IP module's open and close routines.

```
netstack_t *netstack_get_current(void);
netstack_t *netstack_find_by_cred(const cred_t *);
netstack_t *netstack_find_by_stackid(netstackid_t);
void netstack_hold(netstack_t *);
void netstack_rele(netstack_t *);
zoneid_t netstackid_to_zoneid(netstackid_t);
netstackid_t zoneid_to_netstackid(zoneid_t); /* XXX useful? */
```

The current netstack implementation has the netstackid as the same as the zoneid, but the implementation is structured so that the TCP/IP modules do not embed that knowledge but instead use the above two functions.

For the TCP/IP kstats we want to make them available to all the zones that use the same stack. That is the kstats for the shared stack should be visible in all the zones that use the shared stack, while the kstats for an exclusive stack should only be visible in the corresponding zone. The following two functions makes this trivial for the TCP/IP modules; they can just use kstat_create_netstack() instead of kstat_create() or kstat_create_zone():

```
kstat_t *kstat_create_netstack(char *, int, char *, char *, uchar_t,
 uint_t, uchar_t, netstackid_t);
void    kstat_delete_netstack(kstat_t *, netstackid_t);
```

In some cases, such as IPsec being loaded into the kernel, all the stack instances need to be notified. The following allows a module to walk all the netstacks:

```
typedef         int     netstack_handle_t;
void    netstack_next_init(netstack_handle_t *);
void    netstack_next_fini(netstack_handle_t *);
netstack_t      *netstack_next(netstack_handle_t *);
```

14 Privileges

The project introduces the new PRIV_SYS_IP_CONFIG as a subset of PRIV_SYS_NET_CONFIG. The definition of the two is:

privilege PRIV_SYS_IP_CONFIG

Allows a process to configure a system's network interfaces and routes.

Allows a process to configure TCP/IP network parameters using ndd.

Allows a process access to otherwise restricted TCP/IP information using ndd.

Allows a process to configure IPsec.

Allows a process to pop anchored STREAMs modules with matching zoneid.

privilege PRIV_SYS_NET_CONFIG

Allows all that PRIV_SYS_IP_CONFIG allows.

Allows a process to push the rpcmod STREAMs module.

Allows a process to INSERT/REMOVE STREAMs modules on locations other

than the top of the module stack.

A zone with exclusive stack is granted PRIV_SYS_IP_CONFIG and PRIV_SYS_NET_RAW.

In addition, an exclusive stack zone that is given a network interface (e.g., bge1) is given access to that

/dev node (e.g., /dev/bge1). As a result, a process with sufficient privileges in the zone can snoop on bge1. But without access to /dev/bge0 the zone can not perform any operations (ifconfig plumb or snoop) on bge0.

15 Internal TCP/IP changes (Implementation details)

The general structure of the changes to all the TCP/IP kernel modules consist of

- Defining a foo_stack_t that contains all the previously global data that is modified
- Having the init and fini routines call netstack_register() and netstack_unregister(), respectively
- Writing foo_stack_init() and foo_stack_fini() routines which allocate and initiate (and free) an instance of a foo_stack_t
- Making the per-instance (normally per-queue i.e. referenced by q_ptr) structure have a pointer to the foo_stack_t
- Making the foo_open() routine use netstack_find_by_cred() to get the pointer to the netstack_t and as a result a pointer to the foo_stack_t, so that this can be saved in the per-instance structure
- For modules that use kstat_create, modify them to use kstat_create_netstack() which ensures that the statistics are visible in the zones that use that stack.

All the TCP/IP kernel modules follow the above pattern.

16 References

[1] PSARC 2002/174 Virtualization and Namespace Isolation in Solaris

[2] PSARC 2002/188 Least privilege for Solaris