Subject: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Fri, 22 Jun 2007 19:39:24 GMT
View Forum Message <> Reply to Message

Currently all of the prerequisite work for implementing a network namespace (i.e. virtualization of the network stack with one kernel) has already been merged or is in the process of being merged.

Therefore it is now time for a bit of high level design review of the network namespace work and time to begin sending patches.

-- User space semantics

If you are in a different network namespace it looks like you have a separate independent copy of the network stack.

User visible kernel structures that will appear to be per network namespace include network devices, routing tables, sockets, and netfilter rules.

-- The basic design

There will be a network namespace structure that holds the global variables for a network namespace, making those global variables per network namespace.

One of those per network namespace global variables will be the loopback device. Which means the network namespace a packet resides in can be found simply by examining the network device or the socket the packet is traversing.

Either a pointer to this global structure will be passed into the functions that need to reference per network namespace variables or a structure that is already passed in (such as the network device) will be modified to contain a pointer to the network namespace structure.

Depending upon the data structure it will either be modified to hold a per entry network namespace pointer or it there will be a separate copy per network namespace. For large global data structures like the ipv4 routing cache hash table adding an additional pointer to the entries appears the more reasonable solution.

The initialization and cleanup functions will be refactored into functions that do the work on a network namespace basis and functions that perform truly global initialization and cleanup. And a registration mechanism will be available to register functions that are per network namespace.

It is a namespace so like the other namespaces that have been implemented a clone flag will exist to create the namespace during clone or unshare.

There will be an additional network stack feature that will allow you to migrate network devices between namespaces.

When complete all of the features of the network stack ipv4, ipv6, decnet, sysctls, virtual devices, routing tables, scheduling, ipsec, netfilter, etc should be able to operate in a per network namespace fashion.

--- The implementation plan

The plan for implementing this is to first get network namespace infrastructure merged. So that pieces of the network stack can be made to operate in a per network namespace fashion.

Then the plan is to proceed as if we are doing a global kernel lock removal. For each layer of the networking stack pass down the per network namespace parameter to the functions and modify the functions to verify they are only operating on the initial network namespace. Then one piece at a time update the code to handle working in multiple network namespaces, and push the network namespace information down to the lower levels.

This plan calls for a lot of patches that are essentially noise. But the result is simple and generally obviously correct patches, that can be easily reviewed, and can be safely merged one at a time, and don't impose any additional ongoing maintenance overhead.

In my current proof of concept patchset it takes about 100 patches before ipv4 is up and working.

--- Performance

In initial measurements the only performance overhead we have been able to measure is getting the packet to the network namespace. Going through ethernet bridging or routing seems to trigger copies of the packet that slow things down. When packets go directly to the network namespace no performance penalty has yet been measured.

--- The question

At the design level does this approach sound reasonable?

Eric

p.s. I will follow up shortly with a patch that is one implementation
of the basic network namespace infrastructure. Feel free to cut it
to shreds (as it is likely overkill) but it should help put the pieces
of what I am talking about into perspective.

____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] net: Basic network namespace infrastructure.
Posted by ebiederm on Fri, 22 Jun 2007 21:22:47 GMT

View Forum Message <> Reply to Message

This is the basic infrastructure needed to support network
namespaces. This infrastructure is:
- Registration functions to support initializing per network
  namespace data when a network namespaces is created or destroyed.

- struct net. The network namespace datastructure.
  This structure will grow as variables are made per network
  namespace but this is the minimal starting point.

- Functions to grab a reference to the network namespace.
  I provide both get/put functions that keep a network namespace
  from being freed. And hold/release functions serve as weak references
  and will warn if their count is not zero when the data structure
  is freed. Useful for dealing with more complicated data structures
  like the ipv4 route cache.

- A list of all of the network namespaces so we can iterate over them.

- A slab for the network namespace data structures allowing leaks
  to be spotted.

I have deliberately chosen to not make it possible to compile out the
code as the support for per-network namespace initialization and
uninitialization needs to always be compiled in once code has
started using it (even if we don't have network namespaces,
and because no one has ever measured any performance overhead
specific to network namespace infrastructure. As code
to compile out the network namespace pointers etc is complicated
it is best to avoid that code unless that complexity is justified.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
 include/net/net_namespace.h |   66 ++++++++++
 net/core/Makefile           |    2 +-
 net/core/net_namespace.c    |  291 ++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 358 insertions(+), 1 deletions(-)
 create mode 100644 include/net/net_namespace.h
 create mode 100644 net/core/net_namespace.c

diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
new file mode 100644
index 0000000..c909b3a
--- /dev/null
+++ b/include/net/net_namespace.h
@@ -0,0 +1,66 @@
+/*
+ * Operations on the network namespace
+ */
+#ifndef __NET_NET_NAMESPACE_H
+#define __NET_NET_NAMESPACE_H
+
+#include <asm/atomic.h>
+#include <linux/workqueue.h>
+#include <linux/list.h>
+
+struct net {
+ atomic_t count;  /* To decided when the network namespace
+     * should go
+     */
+ atomic_t use_count; /* For references we destroy on demand */
+ struct list_head list; /* list of network namespace structures */
+ struct work_struct work; /* work struct for freeing */
+};
+
+extern struct net init_net;
+extern struct list_head net_namespace_list;
+
+extern void __put_net(struct net *net);
+
+static inline struct net *get_net(struct net *net)
+{
+ atomic_inc(&net->count);
+ return net;
+}
+
+static inline void put_net(struct net *net)
+{
+ if (atomic_dec_and_test(&net->count))
+  __put_net(net);
+}
```

```
+
+static inline struct net *hold_net(struct net *net)
+{
+ atomic_inc(&net->use_count);
+ return net;
+}
+
+static inline void release_net(struct net *net)
+{
+ atomic_dec(&net->use_count);
+}
+
+extern void net_lock(void);
+extern void net_unlock(void);
+
+#define for_each_net(VAR)    \
+ list_for_each_entry(VAR, &net_namespace_list, list);
+
+
+struct pernet_operations {
+ struct list_head list;
+ int (*init)(struct net *net);
+ void (*exit)(struct net *net);
+};
+
+extern int register_pernet_subsys(struct pernet_operations *);
+extern void unregister_pernet_subsys(struct pernet_operations *);
+extern int register_pernet_device(struct pernet_operations *);
+extern void unregister_pernet_device(struct pernet_operations *);
+
+#endif /* __NET_NET_NAMESPACE_H */
diff --git a/net/core/Makefile b/net/core/Makefile
index 4751613..ea9b3f3 100644
--- a/net/core/Makefile
+++ b/net/core/Makefile
@@ -3,7 +3,7 @@
 #

 obj-y := sock.o request_sock.o skbuff.o iovec.o datagram.o stream.o scm.o \
-  gen_stats.o gen_estimator.o
+  gen_stats.o gen_estimator.o net_namespace.o

 obj-$(CONFIG_SYSCTL) += sysctl_net_core.o

diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
new file mode 100644
index 0000000..397c15f
--- /dev/null
```

```
+++ b/net/core/net_namespace.c
@@ -0,0 +1,291 @@
+#include <linux/workqueue.h>
+#include <linux/rtnetlink.h>
+#include <linux/cache.h>
+#include <linux/slab.h>
+#include <linux/list.h>
+#include <linux/delay.h>
+#include <net/net_namespace.h>
+
+/*
+ * Our network namespace constructor/destructor lists
+ */
+
+static LIST_HEAD(pernet_list);
+static struct list_head *first_device = &pernet_list;
+static DEFINE_MUTEX(net_mutex);
+
+static DEFINE_MUTEX(net_list_mutex);
+LIST_HEAD(net_namespace_list);
+
+static struct kmem_cache *net_cachep;
+
+struct net init_net;
+
+void net_lock(void)
+{
+ mutex_lock(&net_list_mutex);
+}
+
+void net_unlock(void)
+{
+ mutex_unlock(&net_list_mutex);
+}
+
+static struct net *net_alloc(void)
+{
+ return kmem_cache_alloc(net_cachep, GFP_KERNEL);
+}
+
+static void net_free(struct net *net)
+{
+ if (!net)
+  return;
+
+ if (unlikely(atomic_read(&net->use_count) != 0)) {
+  printk(KERN_EMERG "network namespace not free! Usage: %d\n",
+   atomic_read(&net->use_count));
+  return;
+ }
+
+ kmem_cache_free(net_cachep, net);
+}
+
+static void cleanup_net(struct work_struct *work)
+{
+ struct pernet_operations *ops;
+ struct list_head *ptr;
+ struct net *net;
+
+ net = container_of(work, struct net, work);
+
+ mutex_lock(&net_mutex);
+
+ /* Don't let anyone else find us. */
+ net_lock();
+ list_del(&net->list);
+ net_unlock();
+
+ /* Run all of the network namespace exit methods */
+ list_for_each_prev(ptr, &pernet_list) {
+  ops = list_entry(ptr, struct pernet_operations, list);
+  if (ops->exit)
+   ops->exit(net);
+ }
+
+ mutex_unlock(&net_mutex);
+
+ /* Ensure there are no outstanding rcu callbacks using this
+  * network namespace.
+  */
+ rcu_barrier();
+
+ /* Finally it is safe to free my network namespace structure */
+ net_free(net);
+}
+
+
+void __put_net(struct net *net)
+{
+ /* Cleanup the network namespace in process context */
+ INIT_WORK(&net->work, cleanup_net);
+ schedule_work(&net->work);
+}
+EXPORT_SYMBOL_GPL(__put_net);
+
+/*
+ * setup_net runs the initializers for the network namespace object.
+ */
+static int setup_net(struct net *net)
+{
+ /* Must be called with net_mutex held */
+ struct pernet_operations *ops;
+ struct list_head *ptr;
+ int error;
+
+ memset(net, 0, sizeof(struct net));
+ atomic_set(&net->count, 1);
+ atomic_set(&net->use_count, 0);
+
+ error = 0;
+ list_for_each(ptr, &pernet_list) {
+  ops = list_entry(ptr, struct pernet_operations, list);
+  if (ops->init) {
+   error = ops->init(net);
+   if (error < 0)
+    goto out_undo;
+  }
+ }
+out:
+ return error;
+out_undo:
+ /* Walk through the list backwards calling the exit functions
+  * for the pernet modules whose init functions did not fail.
+  */
+ for (ptr = ptr->prev; ptr != &pernet_list; ptr = ptr->prev) {
+  ops = list_entry(ptr, struct pernet_operations, list);
+  if (ops->exit)
+   ops->exit(net);
+ }
+ goto out;
+}
+
+static int __init net_ns_init(void)
+{
+ int err;
+
+ printk(KERN_INFO "net_namespace: %zd bytes\n", sizeof(struct net));
+ net_cachep = kmem_cache_create("net_namespace", sizeof(struct net),
+     SMP_CACHE_BYTES,
+     SLAB_PANIC, NULL, NULL);
+ mutex_lock(&net_mutex);
+ err = setup_net(&init_net);
+
+ net_lock();
+ list_add_tail(&init_net.list, &net_namespace_list);
+ net_unlock();
+
+ mutex_unlock(&net_mutex);
+ if (err)
+  panic("Could not setup the initial network namespace");
+
+ return 0;
+}
+
+pure_initcall(net_ns_init);
+
+static int register_pernet_operations(struct list_head *list,
+          struct pernet_operations *ops)
+{
+ struct net *net, *undo_net;
+ int error;
+
+ error = 0;
+ list_add_tail(&ops->list, list);
+ for_each_net(net) {
+  if (ops->init) {
+   error = ops->init(net);
+   if (error)
+    goto out_undo;
+  }
+ }
+out:
+ return error;
+
+out_undo:
+ /* If I have an error cleanup all namespaces I initialized */
+ list_del(&ops->list);
+ for_each_net(undo_net) {
+  if (undo_net == net)
+   goto undone;
+  if (ops->exit)
+   ops->exit(undo_net);
+ }
+undone:
+ goto out;
+}
+
+static void unregister_pernet_operations(struct pernet_operations *ops)
+{
+ struct net *net;
+
+ list_del(&ops->list);
+ for_each_net(net)
+  if (ops->exit)
+   ops->exit(net);
+}
+
+/**
+ *      register_pernet_subsys - register a network namespace subsystem
+ * @ops:  pernet operations structure for the subsystem
+ *
+ * Register a subsystem which has init and exit functions
+ * that are called when network namespaces are created and
+ * destroyed respectively.
+ *
+ * When registered all network namespace init functions are
+ * called for every existing network namespace.  Allowing kernel
+ * modules to have a race free view of the set of network namespaces.
+ *
+ * When a new network namespace is created all of the init
+ * methods are called in the order in which they were registered.
+ *
+ * When a network namespace is destroyed all of the exit methods
+ * are called in the reverse of the order with which they were
+ * registered.
+ */
+int register_pernet_subsys(struct pernet_operations *ops)
+{
+ int error;
+ mutex_lock(&net_mutex);
+ error =  register_pernet_operations(first_device, ops);
+ mutex_unlock(&net_mutex);
+ return error;
+}
+EXPORT_SYMBOL_GPL(register_pernet_subsys);
+
+/**
+ *      unregister_pernet_subsys - unregister a network namespace subsystem
+ * @ops: pernet operations structure to manipulate
+ *
+ * Remove the pernet operations structure from the list to be
+ * used when network namespaces are created or destoryed.  In
+ * addition run the exit method for all existing network
+ * namespaces.
+ */
+void unregister_pernet_subsys(struct pernet_operations *module)
+{
+ mutex_lock(&net_mutex);
+ unregister_pernet_operations(module);
+ mutex_unlock(&net_mutex);
+}
+EXPORT_SYMBOL_GPL(unregister_pernet_subsys);
+
+/**
+ *      register_pernet_device - register a network namespace device
+ * @ops:  pernet operations structure for the subsystem
+ *
+ * Register a device which has init and exit functions
+ * that are called when network namespaces are created and
+ * destroyed respectively.
+ *
+ * When registered all network namespace init functions are
+ * called for every existing network namespace.  Allowing kernel
+ * modules to have a race free view of the set of network namespaces.
+ *
+ * When a new network namespace is created all of the init
+ * methods are called in the order in which they were registered.
+ *
+ * When a network namespace is destroyed all of the exit methods
+ * are called in the reverse of the order with which they were
+ * registered.
+ */
+int register_pernet_device(struct pernet_operations *ops)
+{
+ int error;
+ mutex_lock(&net_mutex);
+ error = register_pernet_operations(&pernet_list, ops);
+ if (!error && (first_device == &pernet_list))
+  first_device = &ops->list;
+ mutex_unlock(&net_mutex);
+ return error;
+}
+EXPORT_SYMBOL_GPL(register_pernet_device);
+
+/**
+ *      unregister_pernet_device - unregister a network namespace netdevice
+ * @ops: pernet operations structure to manipulate
+ *
+ * Remove the pernet operations structure from the list to be
+ * used when network namespaces are created or destoryed.  In
+ * addition run the exit method for all existing network
+ * namespaces.
+ */
+void unregister_pernet_device(struct pernet_operations *ops)
+{
+ mutex_lock(&net_mutex);
+ if (&ops->list == first_device)
+  first_device = first_device->next;
+ unregister_pernet_operations(ops);
+ mutex_unlock(&net_mutex);
+}
+EXPORT_SYMBOL_GPL(unregister_pernet_device);
--
1.5.1.1.181.g2de0
```

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Patrick McHardy on Sat, 23 Jun 2007 10:40:44 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> -- The basic design
>
> There will be a network namespace structure that holds the global
> variables for a network namespace, making those global variables
> per network namespace.
>
> One of those per network namespace global variables will be the
> loopback device. Which means the network namespace a packet resides
> in can be found simply by examining the network device or the socket
> the packet is traversing.
>
> Either a pointer to this global structure will be passed into
> the functions that need to reference per network namespace variables
> or a structure that is already passed in (such as the network device)
> will be modified to contain a pointer to the network namespace
> structure.

I believe OpenVZ stores the current namespace somewhere global,
which avoids passing the namespace around. Couldn't you do this
as well?

> Depending upon the data structure it will either be modified to hold
> a per entry network namespace pointer or it there will be a separate
> copy per network namespace. For large global data structures like
> the ipv4 routing cache hash table adding an additional pointer to the
> entries appears the more reasonable solution.

So the routing cache is shared between all namespaces?

> --- Performance
>
> In initial measurements the only performance overhead we have been
> able to measure is getting the packet to the network namespace.
> Going through ethernet bridging or routing seems to trigger copies
> of the packet that slow things down. When packets go directly to
> the network namespace no performance penalty has yet been measured.

It would be interesting to find out whats triggering these copies.
Do you have NAT enabled?

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Ben Greear on Sat, 23 Jun 2007 15:20:40 GMT

View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Eric W. Biederman wrote:
>
>> -- The basic design
>>
>> There will be a network namespace structure that holds the global
>> variables for a network namespace, making those global variables
>> per network namespace.
>>
>> One of those per network namespace global variables will be the
>> loopback device. Which means the network namespace a packet resides
>> in can be found simply by examining the network device or the socket
>> the packet is traversing.
>>
>> Either a pointer to this global structure will be passed into
>> the functions that need to reference per network namespace variables
>> or a structure that is already passed in (such as the network device)
>> will be modified to contain a pointer to the network namespace
>> structure.
>>
>
>
> I believe OpenVZ stores the current namespace somewhere global,
> which avoids passing the namespace around. Couldn't you do this
> as well?

>
Will we be able to have a single application be in multiple name-spaces?

Thanks,
Ben

--
Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 17:19:34 GMT

View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

> Eric W. Biederman wrote:
>> -- The basic design
>>
>> There will be a network namespace structure that holds the global
>> variables for a network namespace, making those global variables
>> per network namespace.
>>
>> One of those per network namespace global variables will be the
>> loopback device. Which means the network namespace a packet resides
>> in can be found simply by examining the network device or the socket
>> the packet is traversing.
>>
>> Either a pointer to this global structure will be passed into
>> the functions that need to reference per network namespace variables
>> or a structure that is already passed in (such as the network device)
>> will be modified to contain a pointer to the network namespace
>> structure.
>
>
> I believe OpenVZ stores the current namespace somewhere global,
> which avoids passing the namespace around. Couldn't you do this
> as well?

It sucks. Especially in the corner cases.   Think macvlan
with the real network device in one namespace and the ``vlan''

device in another device.

The implementation of a global is also pretty a little questionable.
Last I looked it didn't work on the transmit path at all and
interesting on the receive path.

Further and fundamentally all a global achieves is removing the need
for the noise patches where you pass the pointer into the various
functions. For long term maintenance it doesn't help anything.

All of the other changes such as messing with the
initialization/cleanup and changing access to access the per network
namespace data structure, and modifying the code partly along the way
to reject working in other non-default network namespaces that are
truly intrusive we both still have to make.

So except as an implementation detail how we pass the per network
namespace pointer is uninteresting.

Currently I am trying for the least clever most straight forward
implementation I can find, that doesn't give us a regression
in network stack performance.

So yes if we want to do passing through a magic per cpu global on
the packet receive path now is the time to decide to do that.
Currently I don't see the advantage in doing that so I'm not
suggesting it.

In general if there are any specific objections people have written
complicated code that allows us to avoid those objections, so it
should just be a matter of dusting those patches off. I would much
rather go with something stupid and simple if people are willing to
merge that however.

>> Depending upon the data structure it will either be modified to hold
>> a per entry network namespace pointer or it there will be a separate
>> copy per network namespace. For large global data structures like
>> the ipv4 routing cache hash table adding an additional pointer to the
>> entries appears the more reasonable solution.
>
>
> So the routing cache is shared between all namespaces?

Yes. Each namespaces has it's own view so semantically it's not
shared. But the initial fan out of the hash table 2M or something
isn't something we want to replicate on a per namespace basis even
assuming the huge page allocations could happen.

So we just tag the entries and add the network namespace as one more
part of the key when doing hash table look ups.

>> --- Performance
>>
>> In initial measurements the only performance overhead we have been
>> able to measure is getting the packet to the network namespace.
>> Going through ethernet bridging or routing seems to trigger copies
>> of the packet that slow things down. When packets go directly to
>> the network namespace no performance penalty has yet been measured.
>
>
> It would be interesting to find out whats triggering these copies.
> Do you have NAT enabled?

I would have to go back and look. There was a skb_cow call someplace
in the routing path. Something else with ipfilter, ethernet bridging.
So yes it is probably interesting to dig into.

So the thread where we dug into this last time to the point of
identifying the problem is here:
https://lists.linux-foundation.org/pipermail/containers/2007-March/004309.html
The problem in the bridging was here:
https://lists.linux-foundation.org/pipermail/containers/2007-March/004336.html
I can't find a good pointer to the bit of discussion that described
the routing. I just remember it was an skb_cow somewhere in the
routing output path, I believe at the point where we write in the new
destination IP. I haven't a clue why the copy was triggering.

Design wise the interesting bit was it nothing was measurable when the
network device was in the network namespace. So adding an extra
pointer parameter to functions and dereferencing the pointer has not
measurably affected performance at this point.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure

Posted by ebiederm on Sat, 23 Jun 2007 17:26:28 GMT
View Forum Message <> Reply to Message

Ben Greear <greearb@candelatech.com> writes:

> Patrick McHardy wrote:
>> Eric W. Biederman wrote:
>>
>>> -- The basic design
>>>
>>> There will be a network namespace structure that holds the global
>>> variables for a network namespace, making those global variables
>>> per network namespace.
>>>
>>> One of those per network namespace global variables will be the
>>> loopback device. Which means the network namespace a packet resides
>>> in can be found simply by examining the network device or the socket
>>> the packet is traversing.
>>>
>>> Either a pointer to this global structure will be passed into
>>> the functions that need to reference per network namespace variables
>>> or a structure that is already passed in (such as the network device)
>>> will be modified to contain a pointer to the network namespace
>>> structure.
>>>
>>
>>
>> I believe OpenVZ stores the current namespace somewhere global,
>> which avoids passing the namespace around. Couldn't you do this
>> as well?
>>
> Will we be able to have a single application be in multiple name-spaces?

A single application certainly.   But then an application can be composed
of multiple processes which can be composed of multiple threads.

In my current patches a single task_struct belongs to a single network
namespace. That namespace is used when creating sockets. The sockets
themselves have a namespace tag and that is used when transmitting
packets, or otherwise operating on the socket.

So if you pass a socket from one process to another you can have
sockets that belong to different network namespaces in a single task.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Stephen Hemminger on Sat, 23 Jun 2007 17:26:40 GMT
View Forum Message <> Reply to Message

On Sat, 23 Jun 2007 08:20:40 -0700
Ben Greear <greearb@candelatech.com> wrote:

> Patrick McHardy wrote:
> > Eric W. Biederman wrote:
> >
> >> -- The basic design
> >>
> >> There will be a network namespace structure that holds the global
> >> variables for a network namespace, making those global variables
> >> per network namespace.
> >>
> >> One of those per network namespace global variables will be the
> >> loopback device. Which means the network namespace a packet resides
> >> in can be found simply by examining the network device or the socket
> >> the packet is traversing.
> >>
> >> Either a pointer to this global structure will be passed into
> >> the functions that need to reference per network namespace variables
> >> or a structure that is already passed in (such as the network device)
> >> will be modified to contain a pointer to the network namespace
> >> structure.
> >>
> >
> >
> > I believe OpenVZ stores the current namespace somewhere global,
> > which avoids passing the namespace around. Couldn't you do this
> > as well?

Maybe the current namespace should be attached to something else
like sysfs root? Having multiple namespace indirection possiblities
leads to interesting cases where current namespace is not correctly
associated with current sysfs tree or current proc tree, ...

> Will we be able to have a single application be in multiple name-spaces?

That would break the whole point of namespaces...

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 17:55:09 GMT
View Forum Message <> Reply to Message

Stephen Hemminger <shemminger@linux-foundation.org> writes:

> On Sat, 23 Jun 2007 08:20:40 -0700
> Ben Greear <greearb@candelatech.com> wrote:
>
>> Patrick McHardy wrote:
>> > Eric W. Biederman wrote:
>> >
>> >> -- The basic design
>> >>
>> >> There will be a network namespace structure that holds the global
>> >> variables for a network namespace, making those global variables
>> >> per network namespace.
>> >>
>> >> One of those per network namespace global variables will be the
>> >> loopback device. Which means the network namespace a packet resides
>> >> in can be found simply by examining the network device or the socket
>> >> the packet is traversing.
>> >>
>> >> Either a pointer to this global structure will be passed into
>> >> the functions that need to reference per network namespace variables
>> >> or a structure that is already passed in (such as the network device)
>> >> will be modified to contain a pointer to the network namespace
>> >> structure.
>> >>
>> >
>> >
>> > I believe OpenVZ stores the current namespace somewhere global,
>> > which avoids passing the namespace around. Couldn't you do this
>> > as well?
>
> Maybe the current namespace should be attached to something else
> like sysfs root? Having multiple namespace indirection possiblities
> leads to interesting cases where current namespace is not correctly
> associated with current sysfs tree or current proc tree, ...

Yes. There are some oddities there.

In my current tree there is code that makes proc and sysfs match the inspecting
process.

I haven't quite solved the inspection problem where we want to look at
the namespace of a different process. But as long as we have clean
code to do the basics that isn't a big leap when we come to it.

I'm not really seeing any problems along this line at this point.

The big problem at this point is code review and merging, and in particular breaking this work up into small enough pieces that they can be digested, successfully code reviewed and merged.

Eric

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Patrick McHardy on Sat, 23 Jun 2007 18:00:45 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>>I believe OpenVZ stores the current namespace somewhere global,
>>which avoids passing the namespace around. Couldn't you do this
>>as well?
>
>
> It sucks. Especially in the corner cases.   Think macvlan
> with the real network device in one namespace and the ``vlan''
> device in another device.
>
> The implementation of a global is also pretty a little questionable.
> Last I looked it didn't work on the transmit path at all and
> interesting on the receive path.
>
> Further and fundamentally all a global achieves is removing the need
> for the noise patches where you pass the pointer into the various
> functions. For long term maintenance it doesn't help anything.
>
> All of the other changes such as messing with the
> initialization/cleanup and changing access to access the per network
> namespace data structure, and modifying the code partly along the way
> to reject working in other non-default network namespaces that are
> truly intrusive we both still have to make.
>
> So except as an implementation detail how we pass the per network
> namespace pointer is uninteresting.
>
> Currently I am trying for the least clever most straight forward
> implementation I can find, that doesn't give us a regression

> in network stack performance.
>
> So yes if we want to do passing through a magic per cpu global on
> the packet receive path now is the time to decide to do that.
> Currently I don't see the advantage in doing that so I'm not
> suggesting it.

I think your approach is fine and is probably a lot easier
to review than using something global.

>>>Depending upon the data structure it will either be modified to hold
>>>a per entry network namespace pointer or it there will be a separate
>>>copy per network namespace. For large global data structures like
>>>the ipv4 routing cache hash table adding an additional pointer to the
>>>entries appears the more reasonable solution.
>>
>>
>>So the routing cache is shared between all namespaces?
>
>
> Yes. Each namespaces has it's own view so semantically it's not
> shared. But the initial fan out of the hash table 2M or something
> isn't something we want to replicate on a per namespace basis even
> assuming the huge page allocations could happen.
>
> So we just tag the entries and add the network namespace as one more
> part of the key when doing hash table look ups.

I can wait for the patches, but I would be interested in how
GC is performed and whether limits can be configured per
namespace.
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 19:08:25 GMT

View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

>>>>Depending upon the data structure it will either be modified to hold
>>>>a per entry network namespace pointer or it there will be a separate
>>>>copy per network namespace. For large global data structures like

>>>>the ipv4 routing cache hash table adding an additional pointer to the
>>>>entries appears the more reasonable solution.
>>>
>>>
>>>So the routing cache is shared between all namespaces?
>>
>>
>> Yes. Each namespaces has it's own view so semantically it's not
>> shared. But the initial fan out of the hash table 2M or something
>> isn't something we want to replicate on a per namespace basis even
>> assuming the huge page allocations could happen.
>>
>> So we just tag the entries and add the network namespace as one more
>> part of the key when doing hash table look ups.
>
>
> I can wait for the patches, but I would be interested in how
> GC is performed and whether limits can be configured per
> namespace.

Currently I believe the gc code is unmodified in my patches.

Currently I have been focusing on the normal semantics and just
making something work in a mergeable fashion.

Limits and the like are comparatively easy to add in after the
rest is working so I haven't been focusing on that.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Ben Greear on Sat, 23 Jun 2007 20:03:05 GMT

View Forum Message <> Reply to Message

Stephen Hemminger wrote:
>
>> Will we be able to have a single application be in multiple name-spaces?
>>
>
> That would break the whole point of namespaces...
>
I was hoping that I could open a socket in one name-space and another in
another name

space, and send traffic between them, within a single application. This
is basically what I can do now with my
send-to-self patch and (for more clever virtual-routing schemes + NAT,
with a conn-track
patch that Patrick cooked up for me). It seems these patches I use are
not acceptable
for merge, so I was hoping name-spaces might work instead.

Thanks,
Ben

--
Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Ben Greear on Sat, 23 Jun 2007 20:09:49 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Ben Greear <greearb@candelatech.com> writes:
>
>
>> Will we be able to have a single application be in multiple name-spaces?
>>
>
> A single application certainly.   But then an application can be composed
> of multiple processes which can be composed of multiple threads.
>
> In my current patches a single task_struct belongs to a single network
> namespace. That namespace is used when creating sockets. The sockets
> themselves have a namespace tag and that is used when transmitting
> packets, or otherwise operating on the socket.
>
> So if you pass a socket from one process to another you can have
> sockets that belong to different network namespaces in a single task.
>
Any chance it could allow one to use a single threaded, single process
and do something like
int fd1 = socket(...., namespace1);
int fd2 = socket(...., namespace2);

Or, maybe a sockopt or similar call to move a socket into a particular
namespace?

I can certainly see it being useful to allow a default name-space per
process, but it would be nice
to also allow explicit assignment of a socket to a name-space for
applications that want to span
a large number of name-spaces.

Thanks,
Ben

--
Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Carl-Daniel Hailfinge on Sat, 23 Jun 2007 20:19:34 GMT

View Forum Message <> Reply to Message

On 23.06.2007 19:19, Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>> Eric W. Biederman wrote:
>
>>> Depending upon the data structure it will either be modified to hold
>>> a per entry network namespace pointer or it there will be a separate
>>> copy per network namespace. For large global data structures like
>>> the ipv4 routing cache hash table adding an additional pointer to the
>>> entries appears the more reasonable solution.
>>
>> So the routing cache is shared between all namespaces?
>
> Yes. Each namespaces has it's own view so semantically it's not
> shared. But the initial fan out of the hash table 2M or something
> isn't something we want to replicate on a per namespace basis even
> assuming the huge page allocations could happen.
>
> So we just tag the entries and add the network namespace as one more
> part of the key when doing hash table look ups.

Can one namespace DoS other namespaces' access to the routing cache?
Two scenarios come to mind:
* provoking hash collisions
* lock contention (sorry, haven't checked whether/how we do locking)

Regards,
Carl-Daniel

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 20:31:52 GMT

View Forum Message <> Reply to Message

Carl-Daniel Hailfinger <c-d.hailfinger.devel.2006@gmx.net> writes:

> Can one namespace DoS other namespaces' access to the routing cache?
> Two scenarios come to mind:
> * provoking hash collisions
> * lock contention (sorry, haven't checked whether/how we do locking)

My initial expectation is that the protections we have to prevent one user from performing a DoS on another user generally cover the cases between namespaces as well.

Further in general global caches and global resource management is more efficient then per namespace management.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 20:39:13 GMT

View Forum Message <> Reply to Message

Ben Greear <greearb@candelatech.com> writes:

> Any chance it could allow one to use a single threaded, single process and do
> something like

> int fd1 = socket(...., namespace1);
> int fd2 = socket(...., namespace2);
>
> Or, maybe a sockopt or similar call to move a socket into a particular
> namespace?
>
> I can certainly see it being useful to allow a default name-space per process,
> but it would be nice
> to also allow explicit assignment of a socket to a name-space for applications
> that want to span
> a large number of name-spaces.

That isn't the primary use case so I have not considered it much.
A setsockopt call might be possible.

It is also possible to have a bunch of children opening sockets for you
and passing to the process that wants to do the work. If you have a
sufficiently slow socket creation rate that will not be a problem just
a little cumbersome.

If you can open all of your sockets upfront it is possible to do
something where you open your sockets then unshare your network
namespace and repeat.

I am committed to making general infrastructure not something that is
targeted in a brittle way at only one scenario.

So it may be that we can cover your scenario. However it is just
enough off of the beaten path that I'm not going to worry about it
the first time through. It looks like it is a very small step from
where I am at to where you want to be. So you may be able to cook
up something that will satisfy your requirements relatively easily.

Eric

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Ben Greear on Sat, 23 Jun 2007 20:44:56 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> So it may be that we can cover your scenario. However it is just
> enough off of the beaten path that I'm not going to worry about it
> the first time through. It looks like it is a very small step from

> where I am at to where you want to be. So you may be able to cook
> up something that will satisfy your requirements relatively easily.
>
That sounds fair to me.

I will assume that as long as you can migrate sockets with the methods you
described, it should not be that difficult to do the same with a sockopt
or similar.

I'll revisit this when your patches are in mainline.

Thanks,
Ben

--
Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by davem on Sat, 23 Jun 2007 20:57:37 GMT

View Forum Message <> Reply to Message

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Sat, 23 Jun 2007 11:19:34 -0600

> Further and fundamentally all a global achieves is removing the need
> for the noise patches where you pass the pointer into the various
> functions. For long term maintenance it doesn't help anything.

I don't accept that we have to add another function argument
to a bunch of core routines just to support this crap,
especially since you give no way to turn it off and get
that function argument slot back.

To be honest I think this form of virtualization is a complete
waste of time, even the openvz approach.

We're protecting the kernel from itself, and that's an endless
uphill battle that you will never win. Let's do this kind of
stuff properly with a real minimal hypervisor, hopefully with
appropriate hardware level support and good virtualized device

interfaces, instead of this namespace stuff.

At least the hypervisor approach you have some chance to fully
harden in some verifyable and truly protected way, with
namespaces it's just a pipe dream and everyone who works on
these namespace approaches knows that very well.

The only positive thing that came out of this work is the
great auditing that the openvz folks have done and the bugs
they have found, but it basically ends right there.

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 21:41:16 GMT

View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sat, 23 Jun 2007 11:19:34 -0600
>
>> Further and fundamentally all a global achieves is removing the need
>> for the noise patches where you pass the pointer into the various
>> functions. For long term maintenance it doesn't help anything.
>
> I don't accept that we have to add another function argument
> to a bunch of core routines just to support this crap,
> especially since you give no way to turn it off and get
> that function argument slot back.
>
> To be honest I think this form of virtualization is a complete
> waste of time, even the openvz approach.
>
> We're protecting the kernel from itself, and that's an endless
> uphill battle that you will never win. Let's do this kind of
> stuff properly with a real minimal hypervisor, hopefully with
> appropriate hardware level support and good virtualized device
> interfaces, instead of this namespace stuff.
>
> At least the hypervisor approach you have some chance to fully
> harden in some verifyable and truly protected way, with
> namespaces it's just a pipe dream and everyone who works on
> these namespace approaches knows that very well.
>

> The only positive thing that came out of this work is the
> great auditing that the openvz folks have done and the bugs
> they have found, but it basically ends right there.

Dave thank you for your candor, it looks like I have finally made
the pieces small enough that we can discuss them.

If you want the argument to compile out. That is not a problem at all.
I dropped that part from my patch because it makes infrastructure more
complicated and there appeared to be no gain. However having a type
that you can pass that the compiler can optimize away is not a
problem. Basically you just make the argument:

```
typedef struct {} you_can_compile_me_out;  /* when you don't want it. */
typedef void * you_can_compile_me_out;     /* when you do want it. */
```

And gcc will generate no code to pass the argument when you compile
it out.

As far as the hardening goes. There is definitely a point there,
short of a kernel proof subsystem that sounds correct to me.

There are some other factors that make a different tradeoff interesting.
First hypervisors do not allow global optimizations (because of the
better isolation) so have an inherent performance disadvantage.
Something like a 10x scaling penalty from the figures I have seen.

Even more interesting for me is the possibility of unmodified
application migration. Where the limiting factor is that
you cannot reliably restore an application because the global
identifiers are not available.

So yes monolithic kernels may have grown so complex that they cannot
be verified and thus you cannot actually keep untrusted users from
doing bad things to each other with any degree of certainty.

However the interesting cases for me are cases where the users are not
aggressively hostile with each other but being stuck with one set of
global identifiers are a problem.

Eric
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Jeff Garzik on Sat, 23 Jun 2007 22:15:43 GMT

View Forum Message <> Reply to Message

David Miller wrote:

> I don't accept that we have to add another function argument
> to a bunch of core routines just to support this crap,
> especially since you give no way to turn it off and get
> that function argument slot back.
>
> To be honest I think this form of virtualization is a complete
> waste of time, even the openvz approach.
>
> We're protecting the kernel from itself, and that's an endless
> uphill battle that you will never win. Let's do this kind of
> stuff properly with a real minimal hypervisor, hopefully with
> appropriate hardware level support and good virtualized device
> interfaces, instead of this namespace stuff.

Strongly seconded. This containerized virtualization approach just bloats up the kernel for something that is inherently fragile and IMO less secure -- protecting the kernel from itself.

Plenty of other virt approaches don't stir the code like this, while simultaneously providing fewer, more-clean entry points for the virtualization to occur.

And that's speaking WITHOUT my vendor hat on...

Jeff

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 22:56:49 GMT

View Forum Message <> Reply to Message

Jeff Garzik <jeff@garzik.org> writes:

> David Miller wrote:
>> I don't accept that we have to add another function argument
>> to a bunch of core routines just to support this crap,
>> especially since you give no way to turn it off and get

>> that function argument slot back.
>>
>> To be honest I think this form of virtualization is a complete
>> waste of time, even the openvz approach.
>>
>> We're protecting the kernel from itself, and that's an endless
>> uphill battle that you will never win. Let's do this kind of
>> stuff properly with a real minimal hypervisor, hopefully with
>> appropriate hardware level support and good virtualized device
>> interfaces, instead of this namespace stuff.
>
> Strongly seconded. This containerized virtualization approach just bloats up
> the kernel for something that is inherently fragile and IMO less secure --
> protecting the kernel from itself.
>
> Plenty of other virt approaches don't stir the code like this, while
> simultaneously providing fewer, more-clean entry points for the virtualization
> to occur.

Wrong. I really don't want to get into a my virtualization approach is better then yours. But this is flat out wrong.

99% of the changes I'm talking about introducing are just:
- variable
+ ptr->variable

There are more pieces mostly with when we initialize those variables but that is the essence of the change.

And as opposed to other virtualization approaches so far no one has been able to measure the overhead. I suspect there will be a few more cache line misses somewhere but they haven't shown up yet.

If the only use was strong isolation which Dave complains about I would concur that the namespace approach is inappropriate. However there are a lot other uses.

Eric
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by Jeff Garzik on Sun, 24 Jun 2007 01:28:44 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Jeff Garzik <jeff@garzik.org> writes:
>
>> David Miller wrote:
>>> I don't accept that we have to add another function argument
>>> to a bunch of core routines just to support this crap,
>>> especially since you give no way to turn it off and get
>>> that function argument slot back.
>>>
>>> To be honest I think this form of virtualization is a complete
>>> waste of time, even the openvz approach.
>>>
>>> We're protecting the kernel from itself, and that's an endless
>>> uphill battle that you will never win. Let's do this kind of
>>> stuff properly with a real minimal hypervisor, hopefully with
>>> appropriate hardware level support and good virtualized device
>>> interfaces, instead of this namespace stuff.
>> Strongly seconded. This containerized virtualization approach just bloats up
>> the kernel for something that is inherently fragile and IMO less secure --
>> protecting the kernel from itself.
>>
>> Plenty of other virt approaches don't stir the code like this, while
>> simultaneously providing fewer, more-clean entry points for the virtualization
>> to occur.
>
> Wrong. I really don't want to get into a my virtualization approach is better
> then yours. But this is flat out wrong.

> 99% of the changes I'm talking about introducing are just:
> - variable
> + ptr->variable
>
> There are more pieces mostly with when we initialize those variables but
> that is the essence of the change.

You completely dodged the main objection. Which is OK if you are
selling something to marketing departments, but not OK

Containers introduce chroot-jail-like features that give one a false
sense of security, while still requiring one to "poke holes" in the
illusion to get hardware-specific tasks accomplished.

The capable/not-capable model (i.e. superuser / normal user) is _still_
being secured locally, even after decades of work and whitepapers and
audits.

You are drinking Deep Kool-Aid if you think adding containers to the
myriad kernel subsystems does anything besides increasing fragility, and

decreasing security. You are securing in-kernel subsystems against
other in-kernel subsystems. superuser/user model made that difficult
enough... now containers add exponential audit complexity to that. Who
is to say that a local root does not also pierce the container model?

> And as opposed to other virtualization approaches so far no one has been
> able to measure the overhead. I suspect there will be a few more cache
> line misses somewhere but they haven't shown up yet.
>
> If the only use was strong isolation which Dave complains about I would
> concur that the namespace approach is inappropriate. However there are
> a lot other uses.

Sure there are uses. There are uses to putting the X server into the
kernel, too. At some point complexity and featuritis has to take a back
seat to basic sanity.

Jeff

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by davem on Sun, 24 Jun 2007 05:45:11 GMT

View Forum Message <> Reply to Message

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Sat, 23 Jun 2007 15:41:16 -0600

> If you want the argument to compile out. That is not a problem at all.
> I dropped that part from my patch because it makes infrastructure more
> complicated and there appeared to be no gain. However having a type
> that you can pass that the compiler can optimize away is not a
> problem. Basically you just make the argument:
>
> typedef struct {} you_can_compile_me_out; /* when you don't want it. */
> typedef void * you_can_compile_me_out;   /* when you do want it. */
>
> And gcc will generate no code to pass the argument when you compile
> it out.

I don't want to have to see or be aware of the types or the
fact that we support namespaces when I work on the networking

code.

This is why I like the security layer in the kernel we have,
I can disable it and it's completely not there. And I can
be completely ignorant of it's existence when I work on the
networking stack.

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by davem on Sun, 24 Jun 2007 05:48:53 GMT
View Forum Message <> Reply to Message

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Sat, 23 Jun 2007 16:56:49 -0600

> If the only use was strong isolation which Dave complains about I would
> concur that the namespace approach is inappropriate. However there are
> a lot other uses.

By your very admission the only appropriate use case is when users
are not "hostile" and can be trusted to some extent.

And that by definition makes it not appropriate for a general purpose
operating system like Linux.

Containers are I believe a step backwards, and we're better than that.

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sun, 24 Jun 2007 12:38:38 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sat, 23 Jun 2007 16:56:49 -0600
>
>> If the only use was strong isolation which Dave complains about I would
>> concur that the namespace approach is inappropriate. However there are

>> a lot other uses.
>
> By your very admission the only appropriate use case is when users
> are not "hostile" and can be trusted to some extent.

Yes. Like all of Linux. Totally hostile antagonistic users that
have millions or billions of dollars to spend better figuring
out how to mess each other up and do bad things to each other should
not be running code on the same machine.

> And that by definition makes it not appropriate for a general purpose
> operating system like Linux.

Not at all. The security should be at least as good as between different
users today, and most likely better.

> Containers are I believe a step backwards, and we're better than that.

I heartily disagree.

I do agree that if someone can write a minimal hypervisor it is possible
to provide stronger guarantees of separation on the same hardware then
it is with linux. But that is because a hypervisor should be such a small
code base one man should be able to prove and audit the thing. Which allows
several independent people to reproduce that effort.

So my real admission is that hypervisor should be able to do much
stronger isolation.

Namespaces come in when you want to do silly little things like share the
sysadmin responsibility, or not modify applications that make silly assumptions
or don't want to take the huge resource consumption hit hypervisors require.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sun, 24 Jun 2007 12:58:54 GMT

View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sat, 23 Jun 2007 15:41:16 -0600

>
>> If you want the argument to compile out. That is not a problem at all.
>> I dropped that part from my patch because it makes infrastructure more
>> complicated and there appeared to be no gain. However having a type
>> that you can pass that the compiler can optimize away is not a
>> problem. Basically you just make the argument:
>>
>> typedef struct {} you_can_compile_me_out; /* when you don't want it. */
>> typedef void * you_can_compile_me_out;   /* when you do want it. */
>>
>> And gcc will generate no code to pass the argument when you compile
>> it out.
>
> I don't want to have to see or be aware of the types or the
> fact that we support namespaces when I work on the networking
> code.
>
> This is why I like the security layer in the kernel we have,
> I can disable it and it's completely not there. And I can
> be completely ignorant of it's existence when I work on the
> networking stack.

That is a mostly reasonable objection. I certainly see minimal
impact to people working on the code long term is an important goal.

My impression of the current linux security subsystem is a non-flexible
brittle iptables that works between processes. I'm rather jaded in
that regard because I have fixed several things and broken the security
subsystem, and people yell. So I'm not certain if it is good for anything
or if we can actually ignore it right now. However for most of networking
we can ignore it.

What makes namespaces interesting to me is that they enable us to do things
that we can't do with linux today. Sorry you seem to have a security
mono-focus so this bears repeating.

I'm not trying to hide network namespaces because I think that is
potentially a much more serious maintenance issue.

During the merge and enabling of this we have to audit everything or
not be responsible programmers. So that is going to be slightly
intrusive.

I am convinced I can keep network namespaces something that is so
trivial and obvious to get right you won't have to pay attention
to them.

Mostly it isn't the core paths in the network that are a challenge to
work with but the little out of the way paths that don't take a
network device or a socket, and things like the initialization code
which is almost never touched.

I don't actually thing the core fast paths of the network stack even
make namespaces visible. I will go back an reexamine that.

You are asking for something much more challenging though. You are
asking for absolutely no foot-print. Something that is a completely a
side-car. Given the thousands of global variables in the networking
code I don't know if that is possible to implement.

Eric

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by davem on Mon, 25 Jun 2007 02:39:56 GMT

View Forum Message <> Reply to Message

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Sun, 24 Jun 2007 06:58:54 -0600

> I am convinced I can keep network namespaces something that is so
> trivial and obvious to get right you won't have to pay attention
> to them.

Ok then, I'll hold you to this when you post the rest of your
implementation :-)

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by serue on Mon, 25 Jun 2007 15:11:51 GMT

View Forum Message <> Reply to Message

Quoting David Miller (davem@davemloft.net):
> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sat, 23 Jun 2007 11:19:34 -0600
>
> > Further and fundamentally all a global achieves is removing the need
> > for the noise patches where you pass the pointer into the various
> > functions. For long term maintenance it doesn't help anything.
>
> I don't accept that we have to add another function argument
> to a bunch of core routines just to support this crap,
> especially since you give no way to turn it off and get
> that function argument slot back.
>
> To be honest I think this form of virtualization is a complete
> waste of time, even the openvz approach.
>
> We're protecting the kernel from itself, and that's an endless
> uphill battle that you will never win. Let's do this kind of

Hi David,

just to be clear this isn't so much about security. Security can be provided using selinux, just as with the userid namespace. But like with the userid namespace, this provides usability for the virtual servers, plus some support for restarting checkpointed applications.

That doesn't attempt to justify the extra argument - if you don't like it, you don't like it :) Just wanted to clarify.

thanks,
-serge

> stuff properly with a real minimal hypervisor, hopefully with
> appropriate hardware level support and good virtualized device
> interfaces, instead of this namespace stuff.
>
> At least the hypervisor approach you have some chance to fully
> harden in some verifyable and truly protected way, with
> namespaces it's just a pipe dream and everyone who works on
> these namespace approaches knows that very well.
>
> The only positive thing that came out of this work is the
> great auditing that the openvz folks have done and the bugs
> they have found, but it basically ends right there.
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by serue on Mon, 25 Jun 2007 15:23:13 GMT

View Forum Message <> Reply to Message

Quoting Jeff Garzik (jeff@garzik.org):
> Eric W. Biederman wrote:
> >Jeff Garzik <jeff@garzik.org> writes:
> >
> >>David Miller wrote:
> >>>I don't accept that we have to add another function argument
> >>>to a bunch of core routines just to support this crap,
> >>>especially since you give no way to turn it off and get
> >>>that function argument slot back.
> >>>
> >>>To be honest I think this form of virtualization is a complete
> >>>waste of time, even the openvz approach.
> >>>
> >>>We're protecting the kernel from itself, and that's an endless
> >>>uphill battle that you will never win. Let's do this kind of
> >>>stuff properly with a real minimal hypervisor, hopefully with
> >>>appropriate hardware level support and good virtualized device
> >>>interfaces, instead of this namespace stuff.
> >>Strongly seconded. This containerized virtualization approach just
> >>bloats up
> >>the kernel for something that is inherently fragile and IMO less secure --
> >>protecting the kernel from itself.
> >>
> >>Plenty of other virt approaches don't stir the code like this, while
> >>simultaneously providing fewer, more-clean entry points for the
> >>virtualization
> >>to occur.
> >
> >Wrong. I really don't want to get into a my virtualization approach is
> >better
> >then yours. But this is flat out wrong.
>
> >99% of the changes I'm talking about introducing are just:
> >- variable
> >+ ptr->variable
> >
> >There are more pieces mostly with when we initialize those variables but
> >that is the essence of the change.

>
> You completely dodged the main objection. Which is OK if you are
> selling something to marketing departments, but not OK
>
> Containers introduce chroot-jail-like features that give one a false
> sense of security, while still requiring one to "poke holes" in the
> illusion to get hardware-specific tasks accomplished.
>
> The capable/not-capable model (i.e. superuser / normal user) is _still_
> being secured locally, even after decades of work and whitepapers and
> audits.
>
> You are drinking Deep Kool-Aid if you think adding containers to the
> myriad kernel subsystems does anything besides increasing fragility, and
> decreasing security. You are securing in-kernel subsystems against
> other in-kernel subsystems.

No we're not. As the name 'network namespaces' implies, we are
introducing namespaces for network-related variables. That's it.

We are not trying to protect in-kernel subsystems from each other. In
fact we're not even trying to protect userspace process from each other.
Though that will in part come free when user processes can't access each
other's data because they are in different namespaces. But using an
LSM like selinux or a custom one to tag and enforce isolation would
still be encouraged.

> superuser/user model made that difficult
> enough... now containers add exponential audit complexity to that. Who
> is to say that a local root does not also pierce the container model?

At the moment it does.

> >And as opposed to other virtualization approaches so far no one has been
> >able to measure the overhead. I suspect there will be a few more cache
> >line misses somewhere but they haven't shown up yet.
> >
> >If the only use was strong isolation which Dave complains about I would
> >concur that the namespace approach is inappropriate. However there are
> >a lot other uses.
>
> Sure there are uses. There are uses to putting the X server into the
> kernel, too. At some point complexity and featuritis has to take a back
> seat to basic sanity.

Generally true, yes.

-serge

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Tue, 26 Jun 2007 15:32:15 GMT

View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sun, 24 Jun 2007 06:58:54 -0600
>
>> I am convinced I can keep network namespaces something that is so
>> trivial and obvious to get right you won't have to pay attention
>> to them.
>
> Ok then, I'll hold you to this when you post the rest of your
> implementation :-)

Sounds fair to me. I definitely don't want a network stack
that is noticeably harder to maintain.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by dev on Wed, 27 Jun 2007 14:39:54 GMT

View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Eric W. Biederman wrote:
>
>>-- The basic design
>>
>>There will be a network namespace structure that holds the global
>>variables for a network namespace, making those global variables
>>per network namespace.
>>
>>One of those per network namespace global variables will be the
>>loopback device. Which means the network namespace a packet resides

>>in can be found simply by examining the network device or the socket
>>the packet is traversing.
>>
>>Either a pointer to this global structure will be passed into
>>the functions that need to reference per network namespace variables
>>or a structure that is already passed in (such as the network device)
>>will be modified to contain a pointer to the network namespace
>>structure.
>
>
>
> I believe OpenVZ stores the current namespace somewhere global,
> which avoids passing the namespace around. Couldn't you do this
> as well?

yes, we store a global namespace context on current
(can be stored in per-cpu as well).

do you prefer this way?

Thanks,
Kirill

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by dev on Wed, 27 Jun 2007 14:41:25 GMT

View Forum Message <> Reply to Message

Ben Greear wrote:
> Patrick McHardy wrote:
>
>>Eric W. Biederman wrote:
>>
>>
>>>-- The basic design
>>>
>>>There will be a network namespace structure that holds the global
>>>variables for a network namespace, making those global variables
>>>per network namespace.
>>>
>>>One of those per network namespace global variables will be the
>>>loopback device. Which means the network namespace a packet resides
>>>in can be found simply by examining the network device or the socket

>>>the packet is traversing.
>>>
>>>Either a pointer to this global structure will be passed into
>>>the functions that need to reference per network namespace variables
>>>or a structure that is already passed in (such as the network device)
>>>will be modified to contain a pointer to the network namespace
>>>structure.
>>>
>>
>>
>>I believe OpenVZ stores the current namespace somewhere global,
>>which avoids passing the namespace around. Couldn't you do this
>>as well?
>>
>
> Will we be able to have a single application be in multiple name-spaces?

Application itself can't be in multiple namespaces in both approaches.
But the objects which belong to the application (e.g. sockets opened by an app)
can belong to different namespaces.

Thanks,
Kirill

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by Patrick McHardy on Wed, 27 Jun 2007 14:45:33 GMT

View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Patrick McHardy wrote:
>
>>
>>
>> I believe OpenVZ stores the current namespace somewhere global,
>> which avoids passing the namespace around. Couldn't you do this
>> as well?
>>
>
> yes, we store a global namespace context on current
> (can be stored in per-cpu as well).
>
> do you prefer this way?

I'm not sure, it would avoid passing it around everywhere, but I'd
expect it to be harder to review because you can't simply follow it
as it gets passed around.

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by Ben Greear on Wed, 27 Jun 2007 14:56:15 GMT

View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Patrick McHardy wrote:
>
>> I believe OpenVZ stores the current namespace somewhere global,
>> which avoids passing the namespace around. Couldn't you do this
>> as well?
>>
>
> yes, we store a global namespace context on current
> (can be stored in per-cpu as well).
>
> do you prefer this way?
>
For what it's worth, I don't prefer this way as I can see wanting to
have one application
use several namespaces at once....

Thanks,
Ben

> Thanks,
> Kirill
>
> -
> To unsubscribe from this list: send the line "unsubscribe netdev" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
>

--
Ben Greear <greearb@candelatech.com>

Candela Technologies Inc http://www.candelatech.com

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by dev on Wed, 27 Jun 2007 15:38:35 GMT

View Forum Message <> Reply to Message

Jeff Garzik wrote:
> Eric W. Biederman wrote:
>
>>Jeff Garzik <jeff@garzik.org> writes:
>>
>>
>>>David Miller wrote:
>>>
>>>>I don't accept that we have to add another function argument
>>>>to a bunch of core routines just to support this crap,
>>>>especially since you give no way to turn it off and get
>>>>that function argument slot back.
>>>>
>>>>To be honest I think this form of virtualization is a complete
>>>>waste of time, even the openvz approach.
>>>>
>>>>We're protecting the kernel from itself, and that's an endless
>>>>uphill battle that you will never win. Let's do this kind of
>>>>stuff properly with a real minimal hypervisor, hopefully with
>>>>appropriate hardware level support and good virtualized device
>>>>interfaces, instead of this namespace stuff.
>>>
>>>Strongly seconded. This containerized virtualization approach just bloats up
>>>the kernel for something that is inherently fragile and IMO less secure --
>>>protecting the kernel from itself.
>>>
>>>Plenty of other virt approaches don't stir the code like this, while
>>>simultaneously providing fewer, more-clean entry points for the virtualization
>>>to occur.
>>
>>Wrong. I really don't want to get into a my virtualization approach is better
>>then yours. But this is flat out wrong.
>
>
>>99% of the changes I'm talking about introducing are just:

>>- variable
>>+ ptr->variable
>>
>>There are more pieces mostly with when we initialize those variables but
>>that is the essence of the change.
>
>
> You completely dodged the main objection. Which is OK if you are
> selling something to marketing departments, but not OK

It is a pure illusion that one kind of virtualization is better then the other one.
Look, maybe *hardware* virtualization and a small size of the hypervisor
make you feel safer, however you totally forget about all the emulation drivers etc.
which can have bugs and security implications as well and maybe triggerable
from inside VM.

> Containers introduce chroot-jail-like features that give one a false
> sense of security, while still requiring one to "poke holes" in the
> illusion to get hardware-specific tasks accomplished.

The concepts of users in any OS (and in Linux in particular) give people
a false sense of security as well!
I know a bunch of examples of how one user can crash/DoS/abuse another one
e.g. on RHEL5 or mainstream.
But no one have problems with that and no one thinks that multiuserness
is a step-backward or maybe someone does?!

Yes, there are always bugs and sometimes security implications,
but people leave fine with it when it is fixed.

> The capable/not-capable model (i.e. superuser / normal user) is _still_
> being secured locally, even after decades of work and whitepapers and
> audits.

> You are drinking Deep Kool-Aid if you think adding containers to the
> myriad kernel subsystems does anything besides increasing fragility, and
> decreasing security. You are securing in-kernel subsystems against
> other in-kernel subsystems. superuser/user model made that difficult
> enough... now containers add exponential audit complexity to that. Who
> is to say that a local root does not also pierce the container model?

containers do the only thing:
make sure that objects from one context are not visible to another one.

If containers are not used - everything returns to the case as it is now, i.e.
everything is global and globally visible and no auditing overhead at all.
So if you are not interested in containers - the code auditing
won't be noticably harder for you.

>>And as opposed to other virtualization approaches so far no one has been
>>able to measure the overhead. I suspect there will be a few more cache
>>line misses somewhere but they haven't shown up yet.
>>
>>If the only use was strong isolation which Dave complains about I would
>>concur that the namespace approach is inappropriate. However there are
>>a lot other uses.
>
>
> Sure there are uses. There are uses to putting the X server into the
> kernel, too. At some point complexity and featuritis has to take a back
> seat to basic sanity.

I agree about sanity, however I totally disagree about complexity
you talk about.
Bugs we face/fix in Linux kernel which are found with the help of
that kind of virtualization makes me believe that Linux kernel
only wins from it.

Thanks,
Kirill

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Re: [RFD] L2 Network namespace infrastructure
Posted by dev on Thu, 28 Jun 2007 13:12:01 GMT
View Forum Message <> Reply to Message

Ben Greear wrote:
> Kirill Korotaev wrote:
>
>>Patrick McHardy wrote:
>>
>>
>>>I believe OpenVZ stores the current namespace somewhere global,
>>>which avoids passing the namespace around. Couldn't you do this
>>>as well?
>>>
>>
>>yes, we store a global namespace context on current
>>(can be stored in per-cpu as well).
>>
>>do you prefer this way?

>>
>
> For what it's worth, I don't prefer this way as I can see wanting to
> have one application
> use several namespaces at once....

As I wrote to you in another email, it's not a problem,
since applications itself do not own network namespace.

It is objects like sockets which are binded to net namespace and thus
you can have sockets from different namespaces in one application
regardless of mechanism of context handling we talk about.

Thanks,
Kirill

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by dev on Thu, 28 Jun 2007 14:53:01 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>
\>>Eric W. Biederman wrote:
>>
>>>-- The basic design
>>>
\>>>There will be a network namespace structure that holds the global
\>>>variables for a network namespace, making those global variables
\>>>per network namespace.
>>>
\>>>One of those per network namespace global variables will be the
\>>>loopback device. Which means the network namespace a packet resides
\>>>in can be found simply by examining the network device or the socket
\>>>the packet is traversing.
>>>
\>>>Either a pointer to this global structure will be passed into
\>>>the functions that need to reference per network namespace variables
\>>>or a structure that is already passed in (such as the network device)
\>>>will be modified to contain a pointer to the network namespace
\>>>structure.

>>
>>
>>I believe OpenVZ stores the current namespace somewhere global,
>>which avoids passing the namespace around. Couldn't you do this
>>as well?
>
>
> It sucks. Especially in the corner cases.   Think macvlan
> with the real network device in one namespace and the ``vlan''
> device in another device.

sorry, what's the problem here? I don't see a single problem here
related to the global current context.

> The implementation of a global is also pretty a little questionable.
> Last I looked it didn't work on the transmit path at all and
> interesting on the receive path.
>
> Further and fundamentally all a global achieves is removing the need
> for the noise patches where you pass the pointer into the various
> functions. For long term maintenance it doesn't help anything.

this is not 100% true.

Having global context also helps:
1. to account for time spent in network processing or some other activity
   (like IRQ handling) to appropriate namespace.

2. it is really usefull and helpfull for functions like printk() with virtualized
   syslog buffer. Otherwise you would need to find a context in every place where
   namespace private message should be printed.

   Actually it is the same natural as the 'current' which is global,
   but could be passed from entry.S instead to every function which needs it.

   OVZ experience just tells me that this helps to avoid all this noise
   and not harder to work with then with 'current'. But it's up to you.

Thanks,
Kirill
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers