Subject: Re: Network virtualization/isolation
Posted by ebiederm on Tue, 28 Nov 2006 21:50:03 GMT
View Forum Message <> Reply to Message

Daniel Lezcano <dlezcano@fr.ibm.com> writes:

> Eric W. Biederman wrote:
>> I do not want to get into a big debate on the merits of various
>> techniques at this time. We seem to be in basic agreement
>> about what we are talking about.
>>
>> There is one thing I think we can all agree upon.
>> - Everything except isolation at the network device/L2 layer, does not
>>   allow guests to have the full power of the linux networking stack.
> Agree.
>>
>> - There has been a demonstrated use for the full power of the linux
>>   networking stack in containers..
> Agree.
>>
>> - There are a set of techniques which look as though they will give
>>   us full speed when we do isolation of the network stack at the
>>   network device/L2 layer.
> Agree.

Herbert Poetzl <herbert@13thfloor.at> writes:
> correct, don't get me wrong, I'm absolutely not against
> layer 2 virtualization, but not at the expense of light-
> weight layer 3 isolation, which _is_ the traditional way
> 'containers' are built (see BSD, solaris ...)

Ok. So on this point we agree. Full isolation at the network device/L2 level
is desirable and no one is opposed to that.

There is however a strong feeling especially for the case of application
containers that something more focused on what a non-privileged process can
use and deal with would be nice. The ``L3'' case.

I agree that has potential but I worry about 2 things.
- Premature optimization.
- A poor choice of semantics.
- Feature creep leading to insane semantics.

I feel there is something in the L3 arguments as well and it sounds
like it would be a good idea to flush out the semantics.

For full network isolation we have the case that every process,

every socket, and every network device belongs to a network namespace.
This is enough to derive the network namespace for all other user
visible data structures, and to a large extent to define their semantics.

We still need a definition of the non-privileged case, that is compatible
with the former definition.

.....

What unprivileged user space gets to manipulate are sockets. So perhaps
we can break our model into a network socket namespace and network device
namespace.

I would define it so that for each socket there is exactly one network
socket namespace. And for each network socket namespace there is exactly
one network device namespace.

The network socket namespace would be concerned with the rules for deciding
which local addresses a socket can connect/accept/bind to.

The network device namespace would be concerned with everything else.

The problem I see are the wild card binds. In general unmodified
server applications want to bind to *:port by default. Running
two such applications on different ip addresses is a problem. Even
if you can configure them not to do that it becomes easy to do that
be default.

There are some interesting flexible cases where we want one
application container to have one port on IP, and a different
application container to have a different port on the same IP.

So we need something flexible and not just based on IP addresses.
I think the right answer here is a netfilter table that defines
what we can accept/bind/connect the socket to.

The tricky part is when do we return -EADDRINUSE.

I think we can specify the rules such that if we conflict with
another socket in the same socket namespace the rules remain
as they are today, and the kernel returns it unconditionally.

I think for cases across network socket namespaces it should
be a matter for the rules, to decide if the connection should
happen and what error code to return if the connection does not
happen.

There is a potential in this to have an ambiguous case where two

applications can be listening for connections on the same socket
on the same port and both will allow the connection. If that
is the case I believe the proper definition is the first socket
that we find that will accept the connection gets the connection.

I believe this is a sufficiently general definition that we can
make it work with network types in the kernel including DECNET, IP,
and IPv6.

The only gain I see for having the socket namespace is socket
collision detection, and a convenient tag to distinguish containers.

I think this set of netfilter rules may be an interesting alternative
to ip connection tracking in the current firewall code.

...

Assuming the above scheme works does that sound about what people
actually want to use?

I think with the appropriate set of rules it provides what is needed
for application migration. I.e. 127.0.0.1 can be filtered so that
you can only connect to sockets in your current container.

It does get a little odd because it does allow for the possibility
that you can have multiple connected sockets with same source ip,
source port, destination ip, destination port. If the rules are
setup appropriately. I don't see that peculiarity being visible on
the outside network so it shouldn't be a problem.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers