Subject: containers access control 'roadmap'
Posted by serue on Thu, 06 Sep 2007 16:55:34 GMT
View Forum Message <> Reply to Message

Roadmap is a bit of an exaggeration, but here is a list of the next bit
of work i expect to do relating to containers and access control. The
list gets more vague toward the end, with the intent of going far enough
ahead to show what the final result would hopefully look like.

Please review and tell me where I'm unclear, inconsistant, glossing over
important details, or completely on drugs.

1. introduce CAP_HOST_ADMIN

acts like a mask. If set, all capabilities apply across
namespaces.

is that ok, or do we insist on duplicates for all caps?

brings us into 64-bit caps, so associated patches come
along

As an example, CAP_DAC_OVERRIDE by itself will mean within
the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN
will override userns equivalence checks.

2. introduce per-process cap_bset

Idea is you can start a container with cap-bset not containing
CAP_HOST_ADMIN, for instance.

As namespaces are fleshed out and proper behavior for
cross-namespace access is figured out (see step 7) I
expect behavior under !CAP_HOST_ADMIN with certain
capabilities will change. I.e. if we get a device
namespace, CAP_MKNOD will be different from
CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
start keeping CAP_MKNOD in their container cap_bsets.

3. audit driver code etc for any and all uid==0 checks. Fix those
immediately to take user namespaces into account.

4. introduce inode->user_ns, as per my previous userns patchset from
April (I guess posted in June, according to:
https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)

For now, enforce roughly the following access checks when
inode->user_ns is set:

```
 if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
  allow
 if current->userns==inode->userns {
  if capable(CAP_DAC_OVERRIDE)
   allow
  if current->uid==inode->i_uid
   allow as owner
  inode->i_uid is in current's keychain
   allow as owner
  uid==inode->i_gid in current's groups
   allow as group
 }
 treat as user 'other' (i.e. usually read-only access)
```

5. Then comes the piece where users can get credentials as users in
   other namespaces to store in their keychain.

6. enforce other userns checks like signaling

7. investigate proper behavior for other cross-namespace capabilities.



---

Subject: Re: containers access control 'roadmap'
Posted by Herbert Poetzl on Thu, 06 Sep 2007 17:10:31 GMT
View Forum Message <> Reply to Message

On Thu, Sep 06, 2007 at 11:55:34AM -0500, Serge E. Hallyn wrote:
> Roadmap is a bit of an exaggeration, but here is a list of the next bit
> of work i expect to do relating to containers and access control.  The
> list gets more vague toward the end, with the intent of going far enough
> ahead to show what the final result would hopefully look like.
>
> Please review and tell me where I'm unclear, inconsistant, glossing over
> important details, or completely on drugs.
>
> 1. introduce CAP_HOST_ADMIN
>
>  acts like a mask.  If set, all capabilities apply across
>  namespaces.
>
>  is that ok, or do we insist on duplicates for all caps?

>
> brings us into 64-bit caps, so associated patches come
> along
>
> As an example, CAP_DAC_OVERRIDE by itself will mean within
> the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN
> will override userns equivalence checks.

what does that mean?
guest spaces need to be limited to a certain (mutable)
subset of capabilities to work properly, please explain
how this relates?

> 2. introduce per-process cap_bset
>
> Idea is you can start a container with cap-bset not containing
> CAP_HOST_ADMIN, for instance.
>
> As namespaces are fleshed out and proper behavior for
> cross-namespace access is figured out (see step 7) I
> expect behavior under !CAP_HOST_ADMIN with certain
> capabilities will change. I.e. if we get a device
> namespace, CAP_MKNOD will be different from
> CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
> start keeping CAP_MKNOD in their container cap_bsets.

doesn't sound like a good idea to me, ignoring caps
or disallowing them seems okay, but changing the meaning
between caps (depending on host or guest space) seems
just wrong ...

> 3. audit driver code etc for any and all uid==0 checks. Fix those
> immediately to take user namespaces into account.

okay, sounds good ...

> 4. introduce inode->user_ns, as per my previous userns patchset from
> April (I guess posted in June, according to:
> https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)
>
> For now, enforce roughly the following access checks when
> inode->user_ns is set:
>
> if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
> allow
> if current->userns==inode->userns {
> if capable(CAP_DAC_OVERRIDE)
> allow

>  if current->uid==inode->i_uid
>   allow as owner
>  inode->i_uid is in current's keychain
>   allow as owner
>  uid==inode->i_gid in current's groups
>   allow as group
> }
> treat as user 'other' (i.e. usually read-only access)

what about inodes belonging to several contexts?
(which is a major resource conserving feature of OS
level isolation)

> 5. Then comes the piece where users can get credentials as users in
>    other namespaces to store in their keychain.

does that make sense? wouldn't it be better to have
the keychains 'per context'?

> 6. enforce other userns checks like signaling
>
> 7. investigate proper behavior for other cross-namespace capabilities.

please elaborate ....



---

Subject: Re: containers access control 'roadmap'
Posted by serue on Thu, 06 Sep 2007 18:26:11 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Thu, Sep 06, 2007 at 11:55:34AM -0500, Serge E. Hallyn wrote:
> > Roadmap is a bit of an exaggeration, but here is a list of the next bit
> > of work i expect to do relating to containers and access control.  The
> > list gets more vague toward the end, with the intent of going far enough
> > ahead to show what the final result would hopefully look like.
> >
> > Please review and tell me where I'm unclear, inconsistant, glossing over
> > important details, or completely on drugs.

Thanks for looking this over, Herbert.

> > 1. introduce CAP_HOST_ADMIN

> >
> >  acts like a mask.  If set, all capabilities apply across
> >  namespaces.
> >
> >  is that ok, or do we insist on duplicates for all caps?
> >
> >  brings us into 64-bit caps, so associated patches come
> >  along
> >
> >  As an example, CAP_DAC_OVERRIDE by itself will mean within
> >  the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN
> >  will override userns equivalence checks.
>
> what does that mean?
> guest spaces need to be limited to a certain (mutable)
> subset of capabilities to work properly, please explain

(note that that muable subset of caps for guest spaces is what item #2,
the per-process cap_bset, implements)

> how this relates?

capabilities will give you privileged access within your own container.
Also having CAP_HOST_ADMIN will mean that the capabilities you have can
also be used against objects in other containers.

Now maybe you prefer a model where a "container" is owned by some user
in some namespaces.  All capabilities apply purely within their own
namespace, and a container owner has full rights to the owned
containers.  That makes container vms more like a qemu vm.

Or maybe I just punt this for now altogether, and we address
cross-namespace privileged access if/when we really need it.

> > 2. introduce per-process cap_bset
> >
> >  Idea is you can start a container with cap-bset not containing
> >  CAP_HOST_ADMIN, for instance.
> >
> >  As namespaces are fleshed out and proper behavior for
> >  cross-namespace access is figured out (see step 7) I
> >  expect behavior under !CAP_HOST_ADMIN with certain
> >  capabilities will change.  I.e. if we get a device
> >  namespace, CAP_MKNOD will be different from
> >  CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
> >  start keeping CAP_MKNOD in their container cap_bsets.
>
> doesn't sound like a good idea to me, ignoring caps

> or disallowing them seems okay, but changing the meaning
> between caps (depending on host or guest space) seems
> just wrong ...

Ok your 'doesn't sound like a good idea' is to my blabbing though, not
the the per-process cap_bset.  Right?  So you're again objecting to
CAP_HOST_ADMIN, item #1?

> > 3. audit driver code etc for any and all uid==0 checks.  Fix those
> >    immediately to take user namespaces into account.
>
> okay, sounds good ...

Ok maybe i should make that '#1' and get going as it's the least
contraversial :)

Though I think I still prefer to start with #2.

> > 4. introduce inode->user_ns, as per my previous userns patchset from
> >    April (I guess posted in June, according to:
> >    https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)
> >
> >  For now, enforce roughly the following access checks when
> >  inode->user_ns is set:
> >
> >  if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
> >   allow
> >  if current->userns==inode->userns {
> >   if capable(CAP_DAC_OVERRIDE)
> >    allow
> >   if current->uid==inode->i_uid
> >    allow as owner
> >   inode->i_uid is in current's keychain
> >    allow as owner
> >   uid==inode->i_gid in current's groups
> >    allow as group
> >  }
> >  treat as user 'other' (i.e. usually read-only access)
>
> what about inodes belonging to several contexts?

There's no such thing in the way I was envisioning it.

An inode belongs to one context.  A user can belong to several.

> (which is a major resource conserving feature of OS
> level isolation)

Sure.  Let's say you want to share /usr among many servers.  It exists
in the host user namespace.  In guest user namespaces, anyone including
root will have access to them as though they were user 'other', i.e. if
a directory has 751 perms, you'll get '1'.

> > 5. Then comes the piece where users can get credentials as users in
> >    other namespaces to store in their keychain.
>
> does that make sense? wouldn't it be better to have
> the keychains 'per context'?

Either you misunderstood me, or I misunderstand you.

What I am saying is that there is a 'uid' keychain, which holds things
like (usernamespace 3, uid 5), meaning that even though I am uid 1000 in
usernamespace 1, I am allowed access to usernamespace 3 as though I were
uid 5.

I expect the two common use cases of this to be:

 1. uid 5 on the host system created a virtual server, and gives
 himself a (usernamespace 2, uid 0) key so he is root in the
 virtual server without having to enter it.  (Meaning he can
 signal all processes, access all files, etc)

 2. uid 3000 on the host system is given (usernamespace 2, uid 1001)
 in a virtual server so he can access uid 1001's files in the
 virtual server which has usernamespace 2.

> > 6. enforce other userns checks like signaling
> >
> > 7. investigate proper behavior for other cross-namespace capabilities.
>
> please elaborate ....

Just that we need to go through the list of capabilities and consider
what they mean with and without CAP_HOST_ADMIN.  For instance
CAP_IPC_LOCK doesn't really matter for CAP_HOST_ADMIN since the
namespaces prevent you cross-ns access.  Implications for CAP_NET_ADMIN
remain to be seen, when network namespaces are complete.

-serge

Subject: Re: containers access control 'roadmap'
Posted by Herbert Poetzl on Thu, 06 Sep 2007 20:23:31 GMT
View Forum Message <> Reply to Message

On Thu, Sep 06, 2007 at 01:26:11PM -0500, Serge E. Hallyn wrote:
> Quoting Herbert Poetzl (herbert@13thfloor.at):
> > On Thu, Sep 06, 2007 at 11:55:34AM -0500, Serge E. Hallyn wrote:
> > > Roadmap is a bit of an exaggeration, but here is a list of the
> > > next bit of work i expect to do relating to containers and access
> > > control. The list gets more vague toward the end, with the intent
> > > of going far enough ahead to show what the final result would
> > > hopefully look like.
> > >
> > > Please review and tell me where I'm unclear, inconsistant,
> > > glossing over important details, or completely on drugs.
>
> Thanks for looking this over, Herbert.
>
> > > 1. introduce CAP_HOST_ADMIN
> > >
> > >  acts like a mask.  If set, all capabilities apply across
> > >  namespaces.
> > >
> > >  is that ok, or do we insist on duplicates for all caps?
> > >
> > >  brings us into 64-bit caps, so associated patches come
> > >  along
> > >
> > >  As an example, CAP_DAC_OVERRIDE by itself will mean within
> > >  the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN
> > >  will override userns equivalence checks.
> >
> > what does that mean?
> > guest spaces need to be limited to a certain (mutable)
> > subset of capabilities to work properly, please explain
>
> (note that that muable subset of caps for guest spaces is what item
> #2, the per-process cap_bset, implements)

how is per-process supposed to handle things like
suid-root properly?

> > how this relates?
>
> capabilities will give you privileged access within your own
> container. Also having CAP_HOST_ADMIN will mean that the capabilities
> you have can also be used against objects in other containers.

also, please make sure that you extend the capability

set to 64 bit first, as this would be using up the
last capability (which is not a good idea IMHO)

> Now maybe you prefer a model where a "container" is owned by some
> user in some namespaces. All capabilities apply purely within their
> own namespace, and a container owner has full rights to the owned
> containers. That makes container vms more like a qemu vm.
>
> Or maybe I just punt this for now altogether, and we address
> cross-namespace privileged access if/when we really need it.
>
> > > 2. introduce per-process cap_bset
> > >
> > >  Idea is you can start a container with cap-bset not containing
> > >  CAP_HOST_ADMIN, for instance.
> > >
> > >  As namespaces are fleshed out and proper behavior for
> > >  cross-namespace access is figured out (see step 7) I
> > >  expect behavior under !CAP_HOST_ADMIN with certain
> > >  capabilities will change.  I.e. if we get a device
> > >  namespace, CAP_MKNOD will be different from
> > >  CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
> > >  start keeping CAP_MKNOD in their container cap_bsets.
> >
> > doesn't sound like a good idea to me, ignoring caps
> > or disallowing them seems okay, but changing the meaning
> > between caps (depending on host or guest space) seems
> > just wrong ...
>
> Ok your 'doesn't sound like a good idea' is to my blabbing though,
> not the the per-process cap_bset. Right? So you're again objecting
> to CAP_HOST_ADMIN, item #1?

no, actually it is to the idea having capabilities which
mean different things depending on whether they are
available on the host or inside a guest (because that
would mean handling them different in userspace software
and for administration)

> > > 3. audit driver code etc for any and all uid==0 checks.  Fix those
> > >    immediately to take user namespaces into account.
> >
> > okay, sounds good ...
>
> Ok maybe i should make that '#1' and get going as it's the least
> contraversial :)
>
> Though I think I still prefer to start with #2.

>
> > > 4. introduce inode->user_ns, as per my previous userns patchset from
> > >    April (I guess posted in June, according to:
> > >    https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)
> > >
> > >  For now, enforce roughly the following access checks when
> > >  inode->user_ns is set:
> > >
> > >  if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
> > >   allow
> > >  if current->userns==inode->userns {
> > >   if capable(CAP_DAC_OVERRIDE)
> > >    allow
> > >   if current->uid==inode->i_uid
> > >    allow as owner
> > >   inode->i_uid is in current's keychain
> > >    allow as owner
> > >   uid==inode->i_gid in current's groups
> > >    allow as group
> > >  }
> > >  treat as user 'other' (i.e. usually read-only access)
> >
> > what about inodes belonging to several contexts?
>
> There's no such thing in the way I was envisioning it.
>
> An inode belongs to one context.  A user can belong to several.

well, at least in Linux-VServer, inodes are shared
on a per inode basis between guests, which drastically
reduces the memory and disk overhead if you have more
than one guest of similar nature ...

> > (which is a major resource conserving feature of OS
> > level isolation)
>
> Sure. Let's say you want to share /usr among many servers.
> It exists in the host user namespace.
> In guest user namespaces, anyone including  root will have
> access to them as though they were user 'other', i.e.
> if a directory has 751 perms, you'll get '1'.

no, the inodes are shared in a way that the guest has
(almost) full control over them, including copy on
write functionality when inode contents or properties
change (see unification for details)

i.e. for us, the ability to share inodes between

completely different process _and_ user spaces is
essential because of resource consumption.

> > > 5. Then comes the piece where users can get credentials
> > > as users in other namespaces to store in their keychain.
> >
> > does that make sense? wouldn't it be better to have
> > the keychains 'per context'?
>
> Either you misunderstood me, or I misunderstand you.
>
> What I am saying is that there is a 'uid' keychain, which
> holds things like (usernamespace 3, uid 5), meaning that
> even though I am uid 1000 in usernamespace 1, I am allowed
> access to usernamespace 3 as though I were uid 5.
>
> I expect the two common use cases of this to be:
>
>  1. uid 5 on the host system created a virtual server,
>  and gives himself a (usernamespace 2, uid 0) key
>  so he is root in the virtual server without having
>  to enter it.  (Meaning he can signal all processes,
>  access all files, etc)
>
>  2. uid 3000 on the host system is given (usernamespace
>  2, uid 1001) in a virtual server so he can access
>  uid 1001's files in the virtual server which has
>  usernamespace 2.

do you mean files here or actually inodes or both?
why shouldn't the host context be able to access
any of them without acquiring any credentials?

> > > 6. enforce other userns checks like signaling
> > >
> > > 7. investigate proper behavior for other cross-namespace capabilities.
> >
> > please elaborate ....
>
> Just that we need to go through the list of capabilities
> and consider what they mean with and without CAP_HOST_ADMIN.

see 'bad idea' above: I think they should _exactly_
mean the same, inside and outside a guest ...

> For instance CAP_IPC_LOCK doesn't really matter for
> CAP_HOST_ADMIN since the namespaces prevent you cross-ns
> access.

hmm? maybe I am misunderstanding the entire concept
of CAP_HOST_ADMIN here ... maybe an example could help?

TIA,
Herbert

> Implications for CAP_NET_ADMIN remain to be seen,
> when network namespaces are complete.
>
> -serge

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

---

Subject: Re: containers access control 'roadmap'
Posted by serue on Thu, 06 Sep 2007 21:10:45 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Thu, Sep 06, 2007 at 01:26:11PM -0500, Serge E. Hallyn wrote:
> > Quoting Herbert Poetzl (herbert@13thfloor.at):
> > > On Thu, Sep 06, 2007 at 11:55:34AM -0500, Serge E. Hallyn wrote:
> > > > Roadmap is a bit of an exaggeration, but here is a list of the
> > > > next bit of work i expect to do relating to containers and access
> > > > control. The list gets more vague toward the end, with the intent
> > > > of going far enough ahead to show what the final result would
> > > > hopefully look like.
> > > >
> > > > Please review and tell me where I'm unclear, inconsistant,
> > > > glossing over important details, or completely on drugs.
> > >
> > > Thanks for looking this over, Herbert.
> > >
> > > > 1. introduce CAP_HOST_ADMIN
> > > >
> > > >  acts like a mask.  If set, all capabilities apply across
> > > >  namespaces.
> > > >
> > > >  is that ok, or do we insist on duplicates for all caps?
> > > >
> > > >  brings us into 64-bit caps, so associated patches come
> > > >  along
> > > >
> > > >  As an example, CAP_DAC_OVERRIDE by itself will mean within
> > > >  the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN

> > > >  will override userns equivalence checks.
> > >
> > > what does that mean?
> > > guest spaces need to be limited to a certain (mutable)
> > > subset of capabilities to work properly, please explain
> > >
> > (note that that muable subset of caps for guest spaces is what item
> > #2, the per-process cap_bset, implements)
>
> how is per-process supposed to handle things like
> suid-root properly?

Simple.  It's inherited at fork, and you can take caps out but not put
them back in.

> > > how this relates?
> > >
> > capabilities will give you privileged access within your own
> > container. Also having CAP_HOST_ADMIN will mean that the capabilities
> > you have can also be used against objects in other containers.
>
> also, please make sure that you extend the capability
> set to 64 bit first, as this would be using up the
> last capability (which is not a good idea IMHO)

Of course - unless you talk me out of defining the capability :)

> > Now maybe you prefer a model where a "container" is owned by some
> > user in some namespaces. All capabilities apply purely within their
> > own namespace, and a container owner has full rights to the owned
> > containers. That makes container vms more like a qemu vm.
> > >
> > Or maybe I just punt this for now altogether, and we address
> > cross-namespace privileged access if/when we really need it.
> > >
> > > > 2. introduce per-process cap_bset
> > > > >
> > > > >  Idea is you can start a container with cap-bset not containing
> > > > >  CAP_HOST_ADMIN, for instance.
> > > > >
> > > > >  As namespaces are fleshed out and proper behavior for
> > > > >  cross-namespace access is figured out (see step 7) I
> > > > >  expect behavior under !CAP_HOST_ADMIN with certain
> > > > >  capabilities will change.  I.e. if we get a device
> > > > >  namespace, CAP_MKNOD will be different from
> > > > >  CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
> > > > >  start keeping CAP_MKNOD in their container cap_bsets.
> > > >

> > > doesn't sound like a good idea to me, ignoring caps
> > > or disallowing them seems okay, but changing the meaning
> > > between caps (depending on host or guest space) seems
> > > just wrong ...
> >
> > Ok your 'doesn't sound like a good idea' is to my blabbing though,
> > not the the per-process cap_bset. Right? So you're again objecting
> > to CAP_HOST_ADMIN, item #1?
>
> no, actually it is to the idea having capabilities which
> mean different things depending on whether they are
> available on the host or inside a guest (because that
> would mean handling them different in userspace software
> and for administration)

Whoa - no i am not saying caps would be handled differently based on whether you're in a container or not. In fact from what I've introduced there is no such thing as a 'host' or 'admin' container.

Rather, the single capability, CAP_HOST_ADMIN, just means that your capabilities will also apply to actions on objects in namespaces other than your own. If you don't have CAP_HOST_ADMIN, then capabilities will only give you privileged status with respect to objects in your own namespaces.

So in theory you could have a child container where admin has CAP_HOST_ADMIN, while the initial set of namespaces, or what some might be tempted to otherwise call the 'host container', have taken CAP_HOST_ADMIN out of their cap_bset (after spawning off the child container with the CAP_HOST_ADMIN bit in it's cap_bset).

Is that clearer? Is it less objectionable to you?

> > > > 3. audit driver code etc for any and all uid==0 checks. Fix those
> > > > immediately to take user namespaces into account.
> > >
> > > okay, sounds good ...
> >
> > Ok maybe i should make that '#1' and get going as it's the least
> > contraversial :)
> >
> > Though I think I still prefer to start with #2.
> >
> > > > > 4. introduce inode->user_ns, as per my previous userns patchset from
> > > > > April (I guess posted in June, according to:
> > > > > https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)
> > > > >
> > > > For now, enforce roughly the following access checks when

> > > >  inode->user_ns is set:
> > > >
> > > >  if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
> > > >   allow
> > > >  if current->userns==inode->userns {
> > > >   if capable(CAP_DAC_OVERRIDE)
> > > >    allow
> > > >   if current->uid==inode->i_uid
> > > >    allow as owner
> > > >   inode->i_uid is in current's keychain
> > > >    allow as owner
> > > >   uid==inode->i_gid in current's groups
> > > >    allow as group
> > > >  }
> > > >  treat as user 'other' (i.e. usually read-only access)
> > >
> > > what about inodes belonging to several contexts?
> >
> > There's no such thing in the way I was envisioning it.
> >
> > An inode belongs to one context.  A user can belong to several.
>
> well, at least in Linux-VServer, inodes are shared
> on a per inode basis between guests, which drastically
> reduces the memory and disk overhead if you have more
> than one guest of similar nature ...

And I believe the same can be done with what I am suggesting.

> > > (which is a major resource conserving feature of OS
> > > level isolation)
> >
> > Sure. Let's say you want to share /usr among many servers.
> > It exists in the host user namespace.
> > In guest user namespaces, anyone including  root will have
> > access to them as though they were user 'other', i.e.
> > if a directory has 751 perms, you'll get '1'.
>
> no,

Well, yes: I'm describing my proposal :)

> the inodes are shared in a way that the guest has
> (almost) full control over them, including copy on
> write functionality when inode contents or properties
> change (see unification for details)

In my proposal, the assignment of values to inode->userns, and

enforcement, is left to the filesystem.  So a filesystem can be written
that understands and interprets global user ids, or, to mimic what you
have, a simple stackable cow filesystem could be used.

> i.e. for us, the ability to share inodes between
> completely different process _and_ user spaces is
> essential because of resource consumption.
>
> > > > 5. Then comes the piece where users can get credentials
> > > > as users in other namespaces to store in their keychain.
> > >
> > > does that make sense? wouldn't it be better to have
> > > the keychains 'per context'?
> >
> > Either you misunderstood me, or I misunderstand you.
> >
> > What I am saying is that there is a 'uid' keychain, which
> > holds things like (usernamespace 3, uid 5), meaning that
> > even though I am uid 1000 in usernamespace 1, I am allowed
> > access to usernamespace 3 as though I were uid 5.
> >
> > I expect the two common use cases of this to be:
> >
> >  1. uid 5 on the host system created a virtual server,
> >         and gives himself a (usernamespace 2, uid 0) key
> >         so he is root in the virtual server without having
> >         to enter it.  (Meaning he can signal all processes,
> >    access all files, etc)
> >
> >  2. uid 3000 on the host system is given (usernamespace
> >    2, uid 1001) in a virtual server so he can access
> >    uid 1001's files in the virtual server which has
> >    usernamespace 2.
>
> do you mean files here or actually inodes or both?
> why shouldn't the host context be able to access
> any of them without acquiring any credentials?

Because there is no 'host context', just an initial namespace.

> > > > 6. enforce other userns checks like signaling
> > > >
> > > > 7. investigate proper behavior for other cross-namespace capabilities.
> > >
> > > please elaborate ....
> >
> > Just that we need to go through the list of capabilities
> > and consider what they mean with and without CAP_HOST_ADMIN.

>
> see 'bad idea' above: I think they should _exactly_
> mean the same, inside and outside a guest ...

See my explanation above: there is no 'inside and outside a guest'.

There is just 'with our without the CAP_HOST_ADMIN' capability', where
the CAP_HOST_ADMIN can be irrevocably removed from a process tree using
prctl(PR_SET_CAPBSET, new_set).

> > For instance CAP_IPC_LOCK doesn't really matter for
> > CAP_HOST_ADMIN since the namespaces prevent you cross-ns
> > access.
>
> hmm? maybe I am misunderstanding the entire concept
> of CAP_HOST_ADMIN here ... maybe an example could help?

I've obviously botched this so far... Let me whip up some examples of
how it all works together and email those out tomorrow.

thanks,
-serge

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

---

Subject: Re: containers access control 'roadmap'
Posted by serue on Fri, 07 Sep 2007 18:18:42 GMT
View Forum Message <> Reply to Message

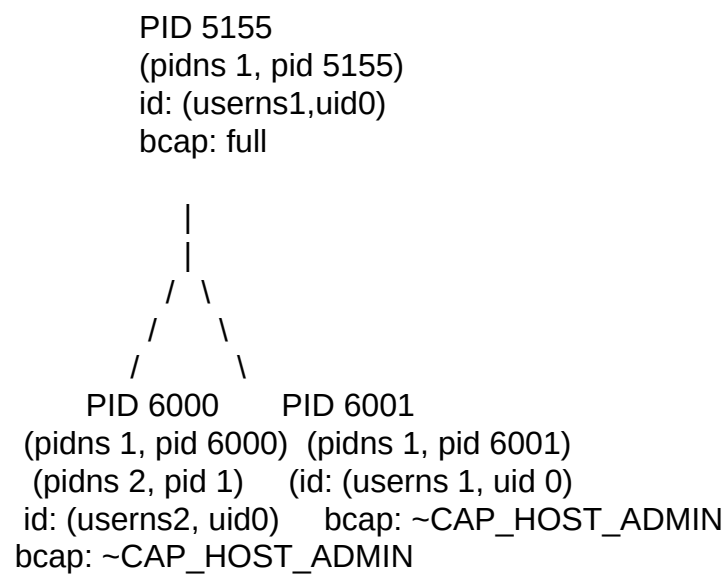
Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Herbert Poetzl (herbert@13thfloor.at):
> > > For instance CAP_IPC_LOCK doesn't really matter for
> > > CAP_HOST_ADMIN since the namespaces prevent you cross-ns
> > > access.
> >
> > hmm? maybe I am misunderstanding the entire concept
> > of CAP_HOST_ADMIN here ... maybe an example could help?
>
> I've obviously botched this so far... Let me whip up some examples of
> how it all works together and email those out tomorrow.
>
> thanks,
> -serge

Ok here is some ranting with an example:

System boots.  All processes have all caps in their cap_bset.
Process 5155 does a clone(CLONE_NEWUSER|CLONE_NEWPID), returning
pid 6000, then does prctl(PR_SET_BCAP, ~CAP_HOST_ADMIN) to
take CAP_HOST_ADMIN out of it's bounding set, meaning it can
never, in any way, gain that capability.

pid 6000 is also (pidns 2, pid 1).  The user owning that
process is (userns 2, uid 0).

Process 5155 does a simple clone(), returning pid 6001, and
that process does prctl(PR_SET_BCAP, ~CAP_HOST_ADMIN).




Process 6000 as root owns a file in its own chroot, let's call it
/vm1/foo.  If process 5155 is still owned by root and tries
to access /vm1/foo, then since it has (CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
it will be able to access the file as root.

If process 6001 is still owned by root, it may have CAP_DAC_OVERRIDE, but
doesn' thave CAP_HOST_ADMIN, so can't cross the userns boundary into userns 2.
So it will get the 'other' perms to /vm1/foo.  However CAP_DAC_OVERRIDE
will apply to let it access files in its own user namespace.

Note that if we were talking about 'host' versus 'guests', then
6001 would be a root process in the 'host'.

Note also that if pid 6000 hadn't dropped CAP_HOST_ADMIN, it would
be a 'guest' which was able to access other namespaces as though
it were the 'host' in a host-guest scheme.

When process 6000 access /vm1/foo, it is in the same userns,

and owns the file, so it can access it.  If it does
setuid(1000), then it can only access /vm1/foo if it has
CAP_DAC_OVERRIDE.  It doesn't need CAP_HOST_ADMIN because
it is not trying to cross a user namespace boundary.

(From here on, I'm *really* speculating, pie in the sky)

I've mentioned - and in previous patchsets started to implement -
that the inode would have a userns pointer.  The filesystem
would pick who to assign an inode to - i.e. based on the superblock,
based on who mounted it, based on who created the file, whatever.
And users would get credentials for userids in other namespaces
through their keyrings.

Right now I'm thinking of taking the same idea but more
generally.  Putting the userns in the inode is too restrictive.
For instance a novel filesystem might well want to ignore uids
altogether and use the keyring to determine file access.
So I'm thinking the filesystem both assigns and checks credentials
for an inode.  For starters, ext2 just

1. assigns the user namespace of the process doing the
   mounting to the superblock
   OR
   if so specified at mounttime, assigns no userns so
   that all user namespaces may access the fs.
2. uses the sb->userns to enforce user namespace checks

Then as a next step it can continue to do the above, but also
allow use of credentials.  Maybe the user who created a userns
with a clone(CLONE_NEWUSER) automatically gets a uid=0 credential
for the new user namespace.  Or some other scheme.

Then, we can get into actually storing key hashes in an inode
xattr, and anyone with a key which hashes to the stored hash gets
access.  Or some more cryptographically sound method of doing
that, please don't bother telling me all the ways that particular
example doesn't work :)

-serge

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