Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Oren Laadan on Thu, 13 Mar 2008 20:01:02 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nadia Derbey <Nadia.Derbey@bull.net> writes:
>
>> Eric W. Biederman wrote:
>>> "Nadia Derbey" <Nadia.Derbey@bull.net> writes:
>>>
>>>> A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>
>>>> Oren's suggestion was to force an object's id during its creation, rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>   (see patch #2 for more details).
>>>>
>>>> When one of these files (or both of them) is filled, a structure pointed to
>>>> by the calling task struct is filled with these ids.
>>>>
>>>> Then, when the object is created, the id(s) present in that structure are
>>>> used, instead of the default ones.
>>>> A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>
>>>> Oren's suggestion was to force an object's id during its creation, rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>   (see patch #2 for more details).
>>>>
>>>> When one of these files (or both of them) is filled, a structure pointed to
>>>> by the calling task struct is filled with these ids.
>>>>

>>>> Then, when the object is created, the id(s) present in that structure are
>>>> used, instead of the default ones.
>>>
>>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>>
>>>
>>>> Right the alloc_pidmap() changes will probably be pretty much the same
>>>> no matter how we do set_it(), so it's worth discussing. But I'm
>>>> particularly curious to see what opinions are on the sys_setid().
>>>
>>> A couple of comments. With respect to alloc_pidmap we already have
>>> the necessary controls (a minimum and a maximum) in place for the
>>> allocation. So except for double checking that those controls are exported
>>> in /proc/sys we don't necessarily need to do anything, special.
>>>
>>> Just play games with the minimum pid value before you fork.
>> Excellent idea! It's trus that properly setting things, we can make the loops
>> executed only once in case we want to use a predefined id.
>>
>>> Second at least to get the memory map correct we need additional
>>> kernel support.
>>>
>>>
>>> Third to actually get the values out it appears we need additional kernel
>>> support as well. From my limited playing with these things at least
>>> parts of the code were easier to implement in the kernel. The hoops
>>> you have to go to restore a single process (without threads) are
>>> absolutely horrendous in user space.
>
> I was thinking in particular of the mm setup above.
>
>> Ok, but if we have a process that belongs to nested namepsaces, we have no other
>> choice than provide its upid nrs hierarchy, right?
>
> At least a part of it. You only have to provide as much of the nested
> pid hiearchy as you are restoring from your checkpoint.
>
>> So, I guess it's more the way it is presented that you don't agree with (of
>> course provided that we are in a user space oriented solution)?
>
> Yes. On the kernel side we are essentially fine.
>
>>> So this patchset whether it is setid or setting the id at creation time
>>> seems to be jumping the gun. For some namespaces renames are valid and
>>> we can support them. For other namespaces setting the id is a big no-no,
>>> and possibly even controlling the id at creation time is a problem (at
>>> least in the general sense).
>> I'm sorry but I'm pretty new in this domain, so I don't see what are the

>> namespaces where setting (or pre-setting) the id would be a problem?
>
> pids to some extent as people use them in all kinds of files. Being
> able to force the pid of another process could make a hard to trigger
> security hole with file permissions absolutely trivial to hit.

Since the intent of this mechanism is to allow ckpt/restart, it makes
sense to only allow this operation during restart. For example, in zap,
containers have a state, e.g. running, stopped, ckpt, restart, and this
is only possible in restart state; Furthermore, a container can only be
put in restart state at creation time, and only by root. Of course, you
should only trust that as much as you trust the root :O

>
> Changing the pid on a process would be even worse because then how
> could you send it signals.
>
> I haven't looked close enough to be able to say in situation X it is a
> problem or in situation Y it is clearly not a problem. I just know
> there is a lot that happens with ids and security so we need to tread
> lightly, and carefully.
>
>>> Because if you can easily control the id
>>> you may be able to more easily exploit security holes.
>>>
>>> I'm not at all inclined to make it easy for userspace to rename or set
>>> the id of a new resource unless it already makes sense in that
>>> namespace.
>>>
>>> We need to limit anything in a checkpoint to user space visible
>>> state.
>> OK, but a tasks that belongs to nested pid namespaces is known from other tasks
>> as one of its "intermediate" pids, depending on the namespace level we are
>> considering. So we can consider these "intermediate pids" as user space visible,
>> can't we?
>
> Yes.
>
>> Given the following pid namespaces hierarchy:
>> PNS0 -> PNS1 -> PNS2 -> PNS3 -> PNS4
>> A task that belongs to PSN2 has 3 upid nrs:
>> UP0, UP1, and UP2
>> So:
>> . UP0 can be obtained if we do a "ps -ef" in pid ns #1 (PNS0)
>> . UP1 can be obtained if we do a "ps -ef" in pid ns #1 (PNS1)
>> . UP2 can be obtained if we do a "ps -ef" in pid ns #1 (PNS2)
>>
>> So UP[0-2] are user space visible (again, depending on "where" we are in pid ns

>> hierarchy, aren't they?
>
> Totally.
>
>> Sure, I completely agree with you! So may be the 1st thing to do would be to
>> decide which approach (user space vs kernel) should be adopted for c/r? Sorry if
>> what I'm saying is stupid, but imho a clear answer to this question would make
>> us all go the same direction ==> save time for further investigations.
>
> Agreed, although it isn't quite that cut and dry.
>
> The question is what granularity do we export the checkpoint restore
> functionality with, and do we manage to have it serve additional
> functions as well.
>
> Because ultimately it is user space saying checkpoint and it is user
> space saying restore. Although we need to be careful that there
> are not cheaper migration solutions that we are precluding.
>
> Getting the user space interface correct is going to be the tricky
> and important.
>
>
>
> My inclination is that the cost in migrating a container is the cost
> of moving the data between machines. Which (not counting filesystems)
> would be the anonymous pages and the shared memory areas.
>
> So if we have a checkpoint as a directory.
> We could have files for the data of each shared memory region.
>
> We could have files for the data of each shared anonymous region
> shared between mm's.
>
> We could have ELF core files with extra notes to describe the full
> state of each process.
>
> We could have files describing each sysvipc object (shared memory,
> message queues, and semaphores).
>
> Futexes?
>
> I would suggest a directory for each different kind of file, making
> conflicts easier to avoid, and data types easier to predict.
>
>
> Once we have a least one valid checkpoint format the question becomes
> how do we get the checkpoint out of the kernel, and how do we get

> the checkpoint into the kernel. And especially how do we allow for
> incremental data movement so live migration with minimal interruption
> of service is possible.
>
> This means that we need to preload all of the large memory areas into
> shared memory areas or processes. Everything else file descriptors
> and the like I expect will be sufficiently inexpesive that we can
> treat their recreation as instantaneous.
>
>
> If incremental migration of data was not so useful I would say just
> have a single syscall to dump everything, and another syscall to
> restore everything.
>
> Since incremental migration is so useful. My gut feel is
> checkpointfs or something similar where we can repeated read
> and write a checkpoint before we commit to starting it is useful.
>
> I am welcome to other better suggestions from the experience of the
> people who have implemented this before.
>
>
> A very fine grained user space solution where user space createsn
> each object using the traditional APIs and restores them similarly
> seems harder to implement, harder to get right, and harder to
> maintain. In part because it hides what we are actually trying to do.
>
> Further a fine grained user space approach appears to offer no
> advantages when it comes to restoring a checkpoint and incrementally
> updating it so that the down time between machines is negligible.
>
>>> My inclination is that create with a specified set of ids is the
>>> proper internal kernel API, so we don't have rework things later,
>>> because reworking things seems to be a lot more work.
>> Completely agree with you: the earlier in the object's life we do it, the best
>> it is ;-)
>>
>>> How we want to
>>> export this to user space is another matter.
>>>
>>> One suggestion is to a have /proc or a proc like filesystem that
>>> allows us to read, create, and populate files to see all of the
>>> application state.
>> And that would be quite useful for debugging purpose, as you were saying
>> earlier.
>
> Yes. Part of why I suggested it in that way. Debugging and
> checkpoint/restart have a lot in common. If we can take advantage of

> that it would be cool.
>
> Eric
> ____
> 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