FreeNAS - Bug #7490

grub-install fails if /boot/grub is a nullfs mount

01/13/2015 11:48 AM - Sean Fagan

Status:	Closed: Not To Be Fixed		
Priority:	Important		
Assignee:	Kris Moore		
Category:	Middleware		
Target version:	N/A		
Seen in:	9.3-RELEASE	**Needs Merging:**	Yes
Severity:	New	**Needs Automation:**	No
Reason for Closing:		**Support Suite Ticket:**	n/a
Reason for Blocked:		**Hardware Configuration:**	
Needs QA:	Yes	**ChangeLog Required:**	No
Needs Doc:	Yes		

Description

There should be no reason for this, when you tell it what device to use.

Use case:

```
beadm create new
    beadm mount new /tmp/mnt
    mount -t nullfs /boot/grub /tmp/mnt/boot/grub
    mount -t devfs devfs /tmp/mnt/dev
    chroot /tmp/mnt /usr/local/bin/grub-install --modules="zfs part_gpt" /dev/da0
```

will fail because it cannot find the device for /boot/grub.

This should be fixable in grub itself.

History

#1 - 01/13/2015 11:48 AM - Sean Fagan

- Blocks Bug #7438: Dtrace is unhappy added

#2 - 01/15/2015 12:14 PM - Jordan Hubbard

- Blocks deleted (Bug #7438: Dtrace is unhappy)

#3 - 01/15/2015 12:49 PM - Jordan Hubbard

- Assignee changed from Xin Li to Kris Moore

#4 - 01/15/2015 01:51 PM - Kris Moore

- Status changed from Unscreened to Screened

#5 - 01/15/2015 01:54 PM - Sean Fagan

Note that I am working around this right now by changing the updater code to unmount /boot/grub, and then remount it in the new environment. This makes me very nervous, because a crash at the wrong time could leave /boot/grub as an empty directory in the root pool.

Being able to tell grub-install to use a particular zfs dataset would work as well for us; I don't know if that's possible.

#6 - 01/15/2015 02:12 PM - Kris Moore

I can take a look into the nullfs issue, but I think it may be more complicated than you think. It may be using /boot/grub to determine what disk partition / file-system UUID contains the grub.cfg and modules for embedding, which is something just giving it "/dev/da0" doesn't supply. (Having a nullfs mount instead I'm sure is totally throwing it for a loop)

However, I think this opens up the question of a bigger issue, on why you are using the nullfs mount in the first place. From what we've seen with GRUB you may be opening yourself up to more pain down the road with user/software driven foot-shooting. What I mean is this:

You have three BE's:

1 - PRIMARY BE (The only one which has valid /boot/grub/* contents)
2 - Old BE 1
3 - Old BE 2

For some reason, #1 gets thrashed, or the user decides to boot into #2 or #3, and then removes #1. Well, now you have a situation where grub.cfg are lost, and a reboot or crash will result in a boot-loader failure.

More importantly you may end up with a situation where if you update grub to a new version (with incompatible modules), and remove #1, now none of the other BE's can be booted, because they still have old copies of /boot/grub/{i386-pc|x86_64-efi}/ modules.

We ran into this a while back (Josh Paetzel may recall, it even happened on his laptop, where he nuked #1 manually and ended up not being able to boot)

To prevent this now in PC-BSD we always do the following:

Anytime a new BE is created, grub-install is run, along with grub-mkconfig, and then we copy /boot/grub to **each** dataset, so no matter what strange voodoo the end-user decides to impose upon their system, they always have the most current grub.cfg / modules in **all** their BEs. Even if they booted some rescue disk and mangled BE's, if they have at least 1 left, they can retrieve their last working config.

To do this we have some code in our "restamp-grub" command that does the following:

1. Restamp grub-install onto the ZFS root disks
   update_grub_boot() { # Bail out if GRUB isn't setup as the default BL on this system
   if [! -e "/boot/grub/grub.cfg"] ; then return 0; fi

   ROOTFS=`mount | awk '/ \/ / {print $1}'`
   BEDS="$(echo ${ROOTFS} | awk -F '/' '{print $2}')"
   if ["$BEDS" = "dev"] ; then BEDS="ROOT"; fi

   `.... snip other code to re-run grub-install ....`

   1. Do the copy of config / modules after we run grub-install, which may update modules
      for i in `beadm list -a 2>/dev/null | grep "/${BEDS}/" | awk '{print $1}'`
      do
      if mount | grep -q "$i on / ("; then
      continue
      fi
      echo -e "Copying grub.cfg to $i...\c" >&2
      fMnt="/mnt.$$"
      mkdir $fMnt
      if ! mount -t zfs ${i} $fMnt ; then
      echo "WARNING: Failed to update grub.cfg on: ${i}" >&2
      continue
      else # Copy grub config and modules over to old dataset # This is done so that newer grub on boot-sector has # matching modules to load from all BE's
      cp /boot/grub/grub.cfg ${fMnt}/boot/grub/grub.cfg
      if [-d "/boot/grub/i386-pc"] ; then
      rm -rf ${fMnt}/boot/grub/i386-pc
      cp -r /boot/grub/i386-pc ${fMnt}/boot/grub/
      fi
      if [-d "/boot/grub/x86_64-efi"] ; then
      rm -rf ${fMnt}/boot/grub/x86_64-efi
      cp -r /boot/grub/x86_64-efi ${fMnt}/boot/grub/
      fi
      echo -e "done" >&2
      umount -f ${fMnt} 2>/dev/null
      fi
      rmdir ${fMnt} 2>/dev/null
      done
      }

#7 - 01/15/2015 02:29 PM - Sean Fagan

We only have one /boot/grub for all the boot environments; that's why I did the nullfs mount.

> It may be using /boot/grub to determine what disk partition / file-system UUID contains the grub.cfg

I'm sure, from our email discussion, that that is the case. That's why I suggested it would work for us to be able to tell it to use a particular zfs dataset, rather than having it look at /boot/grub. E.g., if I were able to do something like

```
grub-install --modules="zfs part_gpt" --config=freenas-root/grub /dev/${disk}
```

that would suffice for us. (Pick options to make sense, I'm just throwing out an idea.)

#8 - 01/16/2015 09:04 AM - Kris Moore

Ok, I got what you are trying here. You want a single dataset for /boot/grub. Looking into grub-install code now to see how we could handle that.

#9 - 01/16/2015 09:18 AM - Kris Moore

Ok, I found some bits in the grub code on how this works. It may be simpler to work-around though. Instead of doing a nullfs mount can you just unmount /boot/grub from the host and re-mount it in the chroot? Even if the system crashes during that process, nothing has changed on the dataset that would render it un-bootable.

1. beadm create new
2. beadm mount new /tmp/mnt
3. umount /boot/grub
4. mount -t zfs freenas-root/grub /tmp/mnt/boot/grub
5. mount -t devfs devfs /tmp/mnt/dev
6. chroot /tmp/mnt /usr/local/bin/grub-install --modules="zfs part_gpt" /dev/da0
7. umount /tmp/mnt/boot/grub
8. umount /tmp/mnt/dev
9. beadm umount new
10. mount -t zfs freenas-root/grub /boot/grub

Let me know if this works on your end. I may change PC-BSD to do something similar, since it would be a bit cleaner to have /boot/grub be a single dataset.

#10 - 01/16/2015 10:24 AM - Sean Fagan

That is in fact what I'm doing now ;). (Except the last one is just "mount /boot/grub" since it's in fstab.)

I was hoping we could repurpose the --boot-directory option to, say, specify zfs:freenas-boot/grub.

#11 - 01/22/2015 10:24 AM - Kris Moore

Status update. I've spent almost two days hacking on this now, killed a few VM's in the process. Making it bypass the nullfs mount isn't so tough, but it breaks the boot in bad ways when it goes to load those modules at boot. Turns out that the '/boot/grub' directory is used for almost all parts of the grub-install process, in figuring out disk devices, uuids, path resolution for modules, etc.

Still hacking on it here, will post back when I get something more tangible for testing.

#12 - 01/22/2015 10:38 AM - Sean Fagan

Don't break your back over it. I'm not happy with the current work-around, but it does work. I had been **assuming** that it would be possible to tell it a specific zfs dataset without mounting that, but if that's not to be, it's not to be. sigh.

#13 - 01/22/2015 11:19 AM - Kris Moore

- Status changed from Screened to Closed: Not To Be Fixed

Yea, so I did more hacking on this, problem is worse than I thought. Grub has to be taught how to bypass all the look-ups of /boot/grub for device name resolution when its on nullfs, in many places.

Not only that, but then it needs a whole new set of flags to grub-install/grub-mkimage/grub-bios-setup telling it when and where to use a specific dataset / zpool for the boot-block embedding for second-stage config loading :/

But now that I understand the issue a bit more, I think it may not be wise to have a single dataset specifically for grub config / modules. Reasoning is this. If something happens in that particular dataset, say a file gets trashed or removed, there is no recourse to boot into another BE. That's it, you are done and need to grab a rescue DVD to try and import your zpool and repair your grub config/files.

If you keep a copy of /boot/grub on each dataset, then if the worst-case happens you can still use the grub-rescue prompts to load the second-stage grub.cfg from another of the BE's /boot/grub directories (which wasn't mounted and ideally not touched by whatever happened to the primary)

Anyway, a specific dataset is your own call for FreeNAS. I would imagine that less users are poking around the shell and tweaking stuff in /boot/grub, whereas I'm got plenty of dual-booters here who mess with grub all the time :P

I'm going to mark this as won't fix for now. If it comes up again in the future we can take another look at it, maybe talk to the grub devs about adding these flags to all the commands upstream first.

#14 - 07/19/2017 09:07 AM - Kris Moore

- Target version changed from Unspecified to N/A