Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Tue, 12 Feb 2008 17:21:34 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> Each new group will have its own maps for char and block
> >> layers. The devices access list is tuned via the
> >> devices.permissions file. One may read from the file to get
> >> the configured state.
> >>
> >> The top container isn't initialized, so that the char
> >> and block layers will use the global maps to lookup
> >> their devices. I did that not to export the static maps
> >> to the outer world.
> >>
> >> Good news is that this patch now contains more comments
> >> and Documentation file :)
> >
> > Seems to work as advertised :) I can't reproduce Suka's null/zero
> > bug.
> >
> > You're relying fully on uid-0 to stop writes into the
> > devices.permissions files. Would you mind adding a check for
> > CAP_SYS_ADMIN (or CAP_NS_OVERRIDE+CAP_MKNOD)? Or were you really
> > counting on using the filesystem visibility cgroup to stop a container
> > from making changes to its device access whitelist?
>
> Yup. I strongly believe that a controller itself should not bring
> any security policy of its own, but the infrastructure should
> take care of this.

That would be ok if the controller gave the infrastructure some way of
knowing what sort of thing the controller does. I.e. I wouldn't mind
having the cgroup infrastructe check for capabilities, but I suspect
some cgroups will really be best represented by different capabilities.

Paul (actually both Menage and Jackson :) do you have an opinion on
this? Are there sites which eg do 'chown -R some_user_id /cgroup/cpusets/'
to have some non-root user be able to dole out cpusets? Is there any
way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?

> However, I'm open to change my mind if I see good explanation of
> why it is wrong.

Well the thing is that it currently leaves no way to keep root in
another namespace from manipulating it. I don't think we can even use

SELinux unless we're willing to prevent containers from having write
access to everything under the cgroup - which is only ok depending on
what is composed with the devices cgroup.

We have the same kind of problem with the
/proc/sys/filesystems/<fs>/fs_safe flag. There are a few possibilities,
and your fs visibility cgroup (plus splitting /proc/sys/filesystems into
its own fs) is one. More complicated things pop into my head, but I
think until we get more comfortable with all this the simplest way is
the best.

But really imo the simplest way is to have a capable(CAP_SYS_ADMIN)
check inside your _write function :)

Ideally if you didn't mind I would float a patch (cousin to the userns
4-patch set) defining CAP_NS_OVERRIDE and requiring both CAP_NS_OVERRIDE
and CAP_SYS_ADMIN to access _write.

-serge

> > thanks,
> > -serge
> >
> >> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> >>
> >> ---
> >>
> >> diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
> >> new file mode 100644
> >> index 0000000..dbd0c7a
> >> --- /dev/null
> >> +++ b/Documentation/controllers/devices.txt
> >> @@ -0,0 +1,61 @@
> >> +
> >> + Devices visibility controller
> >> +
> >> +This controller allows to tune the devices accessibility by tasks,
> >> +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
> >> +access to IDE devices and completely hide SCSI disks.
> >> +
> >> +Tasks still can call mknod to create device files, regardless of
> >> +whether the particular device is visible or accessible, but they
> >> +may not be able to open it later.
> >> +
> >> +This one hides under CONFIG_CGROUP_DEVS option.
> >> +
> >> +
> >> +Configuring

> >> +
> >> +The controller provides a single file to configure itself -- the
> >> +devices.permissions one. To change the accessibility level for some
> >> +device write the following string into it:
> >> +
> >> +[cb] <major>:(<minor>|*) [r-][w-]
> >> + ^ ^ ^
> >> + | | |
> >> + | | +--- access rights (1)
> >> + | |
> >> + | +-- device major and minor numbers (2)
> >> + |
> >> + +-- device type (character / block)
> >> +
> >> +1) The access rights set to '--' remove the device from the group's
> >> +access list, so that it will not even be shown in this file later.
> >> +
> >> +2) Setting the minor to '*' grants access to all the minors for
> >> +particular major.
> >> +
> >> +When reading from it, one may see something like
> >> +
> >> + c 1:5 rw
> >> + b 8:* r-
> >> +
> >> +Security issues, concerning who may grant access to what are governed
> >> +at the cgroup infrastructure level.
> >> +
> >> +
> >> +Examples:
> >> +
> >> +1. Grand full access to /dev/null
> >> + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
> >> +
> >> +2. Grant the read-only access to /dev/sda and partitions
> >> + # echo b 8:* r- > ...
> >> +
> >> +3. Change the /dev/null access to write-only
> >> + # echo c 1:3 -w > ...
> >> +
> >> +4. Revoke access to /dev/sda
> >> + # echo b 8:* -- > ...
> >> +
> >> +
> >> + Written by Pavel Emelyanov <xemul@openvz.org>
> >> +
> >> diff --git a/fs/Makefile b/fs/Makefile
> >> index 7996220..5ad03be 100644

> >> --- a/fs/Makefile
> >> +++ b/fs/Makefile
> >> @@ -64,6 +64,8 @@ obj-y   += devpts/
> >>
> >> obj-$(CONFIG_PROFILING) += dcookies.o
> >> obj-$(CONFIG_DLM) += dlm/
> >> +
> >> +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
> >>
> >> # Do not add any filesystems before this line
> >> obj-$(CONFIG_REISERFS_FS) += reiserfs/
> >> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> >> new file mode 100644
> >> index 0000000..48c5f69
> >> --- /dev/null
> >> +++ b/fs/devscontrol.c
> >> @@ -0,0 +1,314 @@
> >> +/*
> >> + * devscontrol.c - Device Controller
> >> + *
> >> + * Copyright 2007 OpenVZ SWsoft Inc
> >> + * Author: Pavel Emelyanov <xemul at openvz dot org>
> >> + *
> >> + * This program is free software; you can redistribute it and/or modify
> >> + * it under the terms of the GNU General Public License as published by
> >> + * the Free Software Foundation; either version 2 of the License, or
> >> + * (at your option) any later version.
> >> + *
> >> + * This program is distributed in the hope that it will be useful,
> >> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> >> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
> >> + * GNU General Public License for more details.
> >> + */
> >> +
> >> +#include <linux/cgroup.h>
> >> +#include <linux/cdev.h>
> >> +#include <linux/err.h>
> >> +#include <linux/devscontrol.h>
> >> +#include <linux/uaccess.h>
> >> +#include <linux/fs.h>
> >> +#include <linux/genhd.h>
> >> +
> >> +struct devs_cgroup {
> >> + /*
> >> + * The subsys state to build into cgrous infrastructure
> >> + */
> >> + struct cgroup_subsys_state css;
> >> +

```
> >> + /*
> >> +  * The maps of character and block devices. They provide a
> >> +  * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
> >> +  * and block/genhd.c to find out how the ->open() callbacks
> >> +  * work when opening a device.
> >> +  *
> >> +  * Each group will have its onw maps, and at the open()
> >> +  * time code will lookup in this map to get the device
> >> +  * and permissions by its dev_t.
> >> +  */
> >> + struct kobj_map *cdev_map;
> >> + struct kobj_map *bdev_map;
> >> +};
> >> +
> >> +static inline
> >> +struct devs_cgroup *css_to_devs(struct cgroup_subsys_state *css)
> >> +{
> >> + return container_of(css, struct devs_cgroup, css);
> >> +}
> >> +
> >> +static inline
> >> +struct devs_cgroup *cgroup_to_devs(struct cgroup *cont)
> >> +{
> >> + return css_to_devs(cgroup_subsys_state(cont, devs_subsys_id));
> >> +}
> >> +
> >> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
> >> +{
> >> + struct cgroup_subsys_state *css;
> >> +
> >> + css = task_subsys_state(tsk, devs_subsys_id);
> >> + if (css->cgroup->parent == NULL)
> >> +  return NULL;
> >> + else
> >> +  return css_to_devs(css)->cdev_map;
> >> +}
> >> +
> >> +struct kobj_map *task_bdev_map(struct task_struct *tsk)
> >> +{
> >> + struct cgroup_subsys_state *css;
> >> +
> >> + css = task_subsys_state(tsk, devs_subsys_id);
> >> + if (css->cgroup->parent == NULL)
> >> +  return NULL;
> >> + else
> >> +  return css_to_devs(css)->bdev_map;
> >> +}
> >> +
```

```
> >> +static struct cgroup_subsys_state *
> >> +devs_create(struct cgroup_subsys *ss, struct cgroup *cont)
> >> +{
> >> + struct devs_cgroup *devs;
> >> +
> >> + devs = kzalloc(sizeof(struct devs_cgroup), GFP_KERNEL);
> >> + if (devs == NULL)
> >> +  goto out;
> >> +
> >> + devs->cdev_map = cdev_map_init();
> >> + if (devs->cdev_map == NULL)
> >> +  goto out_free;
> >> +
> >> + devs->bdev_map = bdev_map_init();
> >> + if (devs->bdev_map == NULL)
> >> +  goto out_free_cdev;
> >> +
> >> + return &devs->css;
> >> +
> >> +out_free_cdev:
> >> + cdev_map_fini(devs->cdev_map);
> >> +out_free:
> >> + kfree(devs);
> >> +out:
> >> + return ERR_PTR(-ENOMEM);
> >> +}
> >> +
> >> +static void devs_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
> >> +{
> >> + struct devs_cgroup *devs;
> >> +
> >> + devs = cgroup_to_devs(cont);
> >> + bdev_map_fini(devs->bdev_map);
> >> + cdev_map_fini(devs->cdev_map);
> >> + kfree(devs);
> >> +}
> >> +
> >> +/*
> >> + * The devices.permissions file read/write functionality
> >> + *
> >> + * The following two routines parse and print the strings like
> >> + * [cb] <major>:(<minor>|*) [r-][w-]
> >> + */
> >> +
> >> +static int decode_perms_str(char *buf, int *chrdev, dev_t *dev,
> >> +  int *all, mode_t *mode)
> >> +{
> >> + unsigned int major, minor;
```

```
> >> + char *end;
> >> + mode_t tmp;
> >> +
> >> + if (buf[0] == 'c')
> >> +  *chrdev = 1;
> >> + else if (buf[0] == 'b')
> >> +  *chrdev = 0;
> >> + else
> >> +  return -EINVAL;
> >> +
> >> + if (buf[1] != ' ')
> >> +  return -EINVAL;
> >> +
> >> + major = simple_strtoul(buf + 2, &end, 10);
> >> + if (*end != ':')
> >> +  return -EINVAL;
> >> +
> >> + if (end[1] == '*') {
> >> +  if (end[2] != ' ')
> >> +   return -EINVAL;
> >> +
> >> +  *all = 1;
> >> +  minor = 0;
> >> +  end += 2;
> >> + } else {
> >> +  minor = simple_strtoul(end + 1, &end, 10);
> >> +  if (*end != ' ')
> >> +   return -EINVAL;
> >> +
> >> +  *all = 0;
> >> + }
> >> +
> >> + tmp = 0;
> >> +
> >> + if (end[1] == 'r')
> >> +  tmp |= FMODE_READ;
> >> + else if (end[1] != '-')
> >> +  return -EINVAL;
> >> + if (end[2] == 'w')
> >> +  tmp |= FMODE_WRITE;
> >> + else if (end[2] != '-')
> >> +  return -EINVAL;
> >> +
> >> + *dev = MKDEV(major, minor);
> >> + *mode = tmp;
> >> + return 0;
> >> +}
> >> +
```

```
> >> +static int encode_perms_str(char *buf, int len, int chrdev, dev_t dev,
> >> +  int all, mode_t mode)
> >> +{
> >> + int ret;
> >> +
> >> + ret = snprintf(buf, len, "%c %d:", chrdev ? 'c' : 'b', MAJOR(dev));
> >> + if (all)
> >> +  ret += snprintf(buf + ret, len - ret, "*");
> >> + else
> >> +  ret += snprintf(buf + ret, len - ret, "%d", MINOR(dev));
> >> +
> >> + ret += snprintf(buf + ret, len - ret, " %c%c\n",
> >> +   (mode & FMODE_READ) ? 'r' : '-',
> >> +   (mode & FMODE_WRITE) ? 'w' : '-');
> >> +
> >> + return ret + 1;
> >> +}
> >> +
> >> +static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
> >> +  struct file *f, const char __user *ubuf,
> >> +  size_t nbytes, loff_t *pos)
> >> +{
> >> + int err, all, chrdev;
> >> + dev_t dev;
> >> + char buf[64];
> >> + struct devs_cgroup *devs;
> >> + mode_t mode;
> >> +
> >> + if (copy_from_user(buf, ubuf, sizeof(buf)))
> >> +  return -EFAULT;
> >> +
> >> + buf[sizeof(buf) - 1] = 0;
> >> + err = decode_perms_str(buf, &chrdev, &dev, &all, &mode);
> >> + if (err < 0)
> >> +  return err;
> >> +
> >> + devs = cgroup_to_devs(cont);
> >> +
> >> + /*
> >> +  * No locking here is required - all that we need
> >> +  * is provided inside the kobject mapping code
> >> +  */
> >> +
> >> + if (mode == 0) {
> >> +  if (chrdev)
> >> +   err = cdev_del_from_map(devs->cdev_map, dev, all);
> >> +  else
> >> +   err = bdev_del_from_map(devs->bdev_map, dev, all);
```

```
> >> +
> >> +  if (err < 0)
> >> +   return err;
> >> +
> >> +  css_put(&devs->css);
> >> + } else {
> >> +  if (chrdev)
> >> +   err = cdev_add_to_map(devs->cdev_map, dev, all, mode);
> >> +  else
> >> +   err = bdev_add_to_map(devs->bdev_map, dev, all, mode);
> >> +
> >> +  if (err < 0)
> >> +   return err;
> >> +
> >> +  css_get(&devs->css);
> >> + }
> >> +
> >> + return nbytes;
> >> +}
> >> +
> >> +struct devs_dump_arg {
> >> + char *buf;
> >> + int pos;
> >> + int chrdev;
> >> +};
> >> +
> >> +static int devs_dump_one(dev_t dev, int range, mode_t mode, void *x)
> >> +{
> >> + struct devs_dump_arg *arg = x;
> >> + char tmp[64];
> >> + int len;
> >> +
> >> + len = encode_perms_str(tmp, sizeof(tmp), arg->chrdev, dev,
> >> +   range != 1, mode);
> >> +
> >> + if (arg->pos >= PAGE_SIZE - len)
> >> +  return 1;
> >> +
> >> + memcpy(arg->buf + arg->pos, tmp, len);
> >> + arg->pos += len;
> >> + return 0;
> >> +}
> >> +
> >> +static ssize_t devs_read(struct cgroup *cont, struct cftype *cft,
> >> +  struct file *f, char __user *ubuf, size_t nbytes, loff_t *pos)
> >> +{
> >> + struct devs_dump_arg arg;
> >> + struct devs_cgroup *devs;
```

```
> >> + ssize_t ret;
> >> +
> >> + arg.buf = (char *)__get_free_page(GFP_KERNEL);
> >> + if (arg.buf == NULL)
> >> +  return -ENOMEM;
> >> +
> >> + devs = cgroup_to_devs(cont);
> >> + arg.pos = 0;
> >> +
> >> + arg.chrdev = 1;
> >> + cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
> >> +
> >> + arg.chrdev = 0;
> >> + bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);
> >> +
> >> + ret = simple_read_from_buffer(ubuf, nbytes, pos,
> >> +   arg.buf, arg.pos);
> >> +
> >> + free_page((unsigned long)arg.buf);
> >> + return ret;
> >> +}
> >> +
> >> +static struct cftype devs_files[] = {
> >> + {
> >> +  .name = "permissions",
> >> +  .write = devs_write,
> >> +  .read = devs_read,
> >> + },
> >> +};
> >> +
> >> +static int devs_populate(struct cgroup_subsys *ss, struct cgroup *cont)
> >> +{
> >> + return cgroup_add_files(cont, ss,
> >> +   devs_files, ARRAY_SIZE(devs_files));
> >> +}
> >> +
> >> +struct cgroup_subsys devs_subsys = {
> >> + .name = "devices",
> >> + .subsys_id = devs_subsys_id,
> >> + .create = devs_create,
> >> + .destroy = devs_destroy,
> >> + .populate = devs_populate,
> >> +};
> >> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> >> index 228235c..9c0cd2c 100644
> >> --- a/include/linux/cgroup_subsys.h
> >> +++ b/include/linux/cgroup_subsys.h
> >> @@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
```

```
> >>  #endif
> >>
> >>  /* */
> >> +
> >> +#ifdef CONFIG_CGROUP_DEVS
> >> +SUBSYS(devs)
> >> +#endif
> >> +
> >> +/* */
> >> diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
> >> new file mode 100644
> >> index 0000000..38057b9
> >> --- /dev/null
> >> +++ b/include/linux/devscontrol.h
> >> @@ -0,0 +1,26 @@
> >> +#ifndef __DEVS_CONTROL_H__
> >> +#define __DEVS_CONTROL_H__
> >> +struct kobj_map;
> >> +struct task_struct;
> >> +
> >> +/*
> >> + * task_[cb]dev_map - get a map from task. Both calls may return
> >> + * NULL, to indicate, that task doesn't belong to any group and
> >> + * that the global map is to be used.
> >> + */
> >> +
> >> +#ifdef CONFIG_CGROUP_DEVS
> >> +struct kobj_map *task_cdev_map(struct task_struct *);
> >> +struct kobj_map *task_bdev_map(struct task_struct *);
> >> +#else
> >> +static inline kobj_map *task_cdev_map(struct task_struct *tsk)
> >> +{
> >> + return NULL;
> >> +}
> >> +
> >> +static inline kobj_map *task_bdev_map(struct task_struct *tsk)
> >> +{
> >> + return NULL;
> >> +}
> >> +#endif
> >> +#endif
> >> diff --git a/init/Kconfig b/init/Kconfig
> >> index 732a1c2..f9a1b4f 100644
> >> --- a/init/Kconfig
> >> +++ b/init/Kconfig
> >> @@ -283,6 +283,19 @@ config CGROUP_DEBUG
> >>
> >>     Say N if unsure
```

> >>
> >> +config CGROUP_DEVS
> >> + bool "Devices cgroup subsystem"
> >> + depends on CGROUPS
> >> + help
> >> +   Controlls the access rights to devices, i.e. you may hide
> >> +   some of them from tasks, so that they will not be able
> >> +   to open them, or you may grant a read-only access to some
> >> +   of them.
> >> +
> >> +   See Documentation/controllers/devices.txt for details.
> >> +
> >> +   This is harmless to say N here, so do it if unsure.
> >> +
> >> config CGROUP_NS
> >>   bool "Namespace cgroup subsystem"
> >>   depends on CGROUPS
> >

_____

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