Subject: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Mon, 24 Sep 2007 08:28:19 GMT
View Forum Message <> Reply to Message

Hi.

At KS we have pointed out the need in some container, that allows
to limit the visibility of some devices to task within it. I.e.
allow for /dev/null, /dev/zero etc, but disable (by default) some
IDE devices or SCSI discs and so on.

Here's the beta of the container. Currently this only allows to
hide the _character_ devices only from the living tasks. To play
with it you just create the container like this

```
 # mount -t container none /cont/devs -o devices
 # mkdir /cont/devs/0
```

it will have two specific files

```
 # ls /cont/devs
devices.block  devices.char  notify_on_release  releasable  release_agent  tasks
```

then move a task into it

```
 # /bin/echo -n $$ > /cont/devs/0/tasks
```

after this you won't be able to read from even /dev/zero

```
 # hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor
```

meanwhile from another ssh session you will. You may allow access
to /dev/zero like this

```
 # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
```

More generally, the '+<major>:<minor>' string grants access to
some device, and '-<major>:<minor>' disables one.

The TODO list now looks like this:
* add the block devices support :) don't know how to make it yet;
* make /proc/devices show relevant info depending on who is
  reading it. currently even if major 1 is disabled for task,
  it will be listed in this file;
* make it possible to enable/disable not just individual major:minor
  pair, but something more flexible, e.g. major:* for all minors

```
  for given major or major:m1-m2 for minor range, etc;
* add the ability to restrict the read/write permissions for a
  container. currently one may just control the visible-invisible
  state for a device in a container, but maybe just readable or
  just writable would be better.

This patch is minimally tested, because I just want to know your
opinion on whether it worths developing the container in such a way or not.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---

diff --git a/drivers/base/map.c b/drivers/base/map.c
index e87017f..0188053 100644
--- a/drivers/base/map.c
+++ b/drivers/base/map.c
@@ -153,3 +153,21 @@ struct kobj_map *kobj_map_init(kobj_prob
  p->lock = lock;
  return p;
 }
+
+void kobj_map_fini(struct kobj_map *map)
+{
+ int i;
+ struct probe *p, *next;
+
+ for (i = 0; i < 256; i++) {
+  p = map->probes[i];
+  while (p->next != NULL) {
+   next = p->next;
+   kfree(p);
+   p = next;
+  }
+ }
+
+ kfree(p);
+ kfree(map);
+}
diff --git a/fs/Makefile b/fs/Makefile
index 2661ef9..837c731 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_CONTAINER_DEVS) += devscontrol.o

 # Do not add any filesystems before this line
 obj-$(CONFIG_REISERFS_FS) += reiserfs/
diff --git a/fs/char_dev.c b/fs/char_dev.c
index c3bfa76..1b0e4da 100644
--- a/fs/char_dev.c
+++ b/fs/char_dev.c
@@ -22,6 +22,8 @@
 #include <linux/mutex.h>
 #include <linux/backing-dev.h>

+#include <linux/devscontrol.h>
+
 #ifdef CONFIG_KMOD
 #include <linux/kmod.h>
 #endif
@@ -362,17 +364,24 @@ int chrdev_open(struct inode * inode, st
  struct cdev *p;
  struct cdev *new = NULL;
  int ret = 0;
+ struct kobj_map *map;
+
+ map = task_cdev_map(current);
+ if (map == NULL)
+  map = cdev_map;

  spin_lock(&cdev_lock);
  p = inode->i_cdev;
- if (!p) {
+ if (!p || p->last != map) {
   struct kobject *kobj;
   int idx;
+
   spin_unlock(&cdev_lock);
-  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
+  kobj = kobj_lookup(map, inode->i_rdev, &idx);
   if (!kobj)
    return -ENXIO;
   new = container_of(kobj, struct cdev, kobj);
+  BUG_ON(p != NULL && p != new);
   spin_lock(&cdev_lock);
   p = inode->i_cdev;
   if (!p) {
@@ -384,6 +393,8 @@ int chrdev_open(struct inode * inode, st
    ret = -ENXIO;
  } else if (!cdev_get(p))
   ret = -ENXIO;
```

```
+ if (p)
+  p->last = map;
  spin_unlock(&cdev_lock);
  cdev_put(new);
  if (ret)
@@ -461,6 +472,49 @@ int cdev_add(struct cdev *p, dev_t dev,
  return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
 }

+int cdev_add_to_map(struct kobj_map *map, dev_t dev)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ tmp = kobj_map(map, dev, 1, NULL, exact_match, exact_lock, c);
+ if (tmp < 0) {
+  cdev_put(c);
+  return tmp;
+ }
+
+ return 0;
+}
+
+int cdev_del_from_map(struct kobj_map *map, dev_t dev)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ kobj_unmap(map, dev, 1);
+
+ spin_lock(&cdev_lock);
+ if (c->last == map)
+  c->last = NULL;
+ spin_unlock(&cdev_lock);
+
+ cdev_put(c);
```

```
+ cdev_put(c);
+ return 0;
+}
+
 static void cdev_unmap(dev_t dev, unsigned count)
 {
  kobj_unmap(cdev_map, dev, count);
@@ -542,6 +596,16 @@ static struct kobject *base_probe(dev_t
  return NULL;
 }

+struct kobj_map *cdev_map_init(void)
+{
+ return kobj_map_init(base_probe, &chrdevs_lock);
+}
+
+void cdev_map_fini(struct kobj_map *map)
+{
+ kobj_map_fini(map);
+}
+
 void __init chrdev_init(void)
 {
  cdev_map = kobj_map_init(base_probe, &chrdevs_lock);
diff --git a/fs/devscontrol.c b/fs/devscontrol.c
new file mode 100644
index 0000000..6fb5f05
--- /dev/null
+++ b/fs/devscontrol.c
@@ -0,0 +1,170 @@
+/*
+ * devscontrol.c - Device Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul@openvz.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/container.h>
```

```
+#include <linux/cdev.h>
+#include <linux/err.h>
+#include <linux/devscontrol.h>
+#include <linux/uaccess.h>
+
+struct devs_container {
+ struct container_subsys_state css;
+
+ struct kobj_map *cdev_map;
+};
+
+static inline
+struct devs_container *css_to_devs(struct container_subsys_state *css)
+{
+ return container_of(css, struct devs_container, css);
+}
+
+static inline
+struct devs_container *container_to_devs(struct container *cont)
+{
+ return css_to_devs(container_subsys_state(cont, devs_subsys_id));
+}
+
+struct kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ struct container_subsys_state *css;
+
+ css = task_subsys_state(tsk, devs_subsys_id);
+ if (css->container->parent == NULL)
+  return NULL;
+ else
+  return css_to_devs(css)->cdev_map;
+}
+
+static struct container_subsys_state *
+devs_create(struct container_subsys *ss, struct container *cont)
+{
+ struct devs_container *devs;
+
+ devs = kzalloc(sizeof(struct devs_container), GFP_KERNEL);
+ if (devs == NULL)
+  goto out;
+
+ devs->cdev_map = cdev_map_init();
+ if (devs->cdev_map == NULL)
+  goto out_free;
+
+ return &devs->css;
```

```
+
+out_free:
+ kfree(devs);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void devs_destroy(struct container_subsys *ss, struct container *cont)
+{
+ struct devs_container *devs;
+
+ devs = container_to_devs(cont);
+ cdev_map_fini(devs->cdev_map);
+ kfree(devs);
+}
+
+static int decode_dev_name(char *buf, dev_t *dev)
+{
+ unsigned int major, minor;
+ char *end;
+
+ major = simple_strtoul(buf, &end, 10);
+ if (*end != ':')
+  return -EINVAL;
+
+ minor = simple_strtoul(end + 1, &end, 10);
+ if (*end != '\0')
+  return -EINVAL;
+
+ *dev = MKDEV(major, minor);
+ return 0;
+}
+
+static ssize_t
+devs_char_write(struct container *cont, struct cftype *cft, struct file *f,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ int err;
+ dev_t dev;
+ char buf[64];
+ struct devs_container *devs;
+
+ if (copy_from_user(buf, userbuf, sizeof(buf)))
+  return -EFAULT;
+
+ err = decode_dev_name(buf + 1, &dev);
+ if (err < 0)
+  return err;
```

```
+
+ devs = container_to_devs(cont);
+
+ switch (buf[0]) {
+ case '+':
+  err = cdev_add_to_map(devs->cdev_map, dev);
+  if (err < 0)
+   return err;
+
+  css_get(&devs->css);
+  break;
+ case '-':
+  err = cdev_del_from_map(devs->cdev_map, dev);
+  if (err < 0)
+   return err;
+
+  css_put(&devs->css);
+  break;
+ default:
+  return -EINVAL;
+ }
+
+ return nbytes;
+}
+
+static ssize_t
+devs_block_write(struct container *cont, struct cftype *cft, struct file *f,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ return -ENOTTY;
+}
+
+static struct cftype devs_files[] = {
+ {
+  .name = "char",
+  .write = devs_char_write,
+ },
+ {
+  .name = "block",
+  .write = devs_block_write,
+ },
+};
+
+static int devs_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_files(cont, ss,
+   devs_files, ARRAY_SIZE(devs_files));
+}
```

```
+
+struct container_subsys devs_subsys = {
+ .name = "devices",
+ .subsys_id = devs_subsys_id,
+ .create = devs_create,
+ .destroy = devs_destroy,
+ .populate = devs_populate,
+};
diff --git a/include/linux/cdev.h b/include/linux/cdev.h
index 1e29b13..0eded40 100644
--- a/include/linux/cdev.h
+++ b/include/linux/cdev.h
@@ -9,6 +9,7 @@
 struct file_operations;
 struct inode;
 struct module;
+struct kobj_map;

 struct cdev {
  struct kobject kobj;
@@ -17,6 +18,7 @@ struct cdev {
  struct list_head list;
  dev_t dev;
  unsigned int count;
+ struct kobj_map *last;
 };

 void cdev_init(struct cdev *, const struct file_operations *);
@@ -33,5 +35,9 @@ void cd_forget(struct inode *);

 extern struct backing_dev_info directly_mappable_cdev_bdi;

+int cdev_add_to_map(struct kobj_map *map, dev_t dev);
+int cdev_del_from_map(struct kobj_map *map, dev_t dev);
+struct kobj_map *cdev_map_init(void);
+void cdev_map_fini(struct kobj_map *map);
 #endif
 #endif
diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
index 81d11c2..9315a9b 100644
--- a/include/linux/container_subsys.h
+++ b/include/linux/container_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_container)
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_DEVS
```

```
+SUBSYS(devs)
+#endif
+
+/* */
diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
new file mode 100644
index 0000000..51ae916
--- /dev/null
+++ b/include/linux/devscontrol.h
@@ -0,0 +1,14 @@
+#ifndef __DEVS_CONTROL_H__
+#define __DEVS_CONTROL_H__
+struct kobj_map;
+struct task_struct;
+
+#ifdef CONFIG_CONTAINER_DEVS
+struct kobj_map *task_cdev_map(struct task_struct *);
+#else
+static inline kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ return NULL;
+}
+#endif
+#endif
diff --git a/include/linux/kobj_map.h b/include/linux/kobj_map.h
index bafe178..2476f8d 100644
--- a/include/linux/kobj_map.h
+++ b/include/linux/kobj_map.h
@@ -10,5 +10,6 @@ int kobj_map(struct kobj_map *, dev_t, u
 void kobj_unmap(struct kobj_map *, dev_t, unsigned long);
 struct kobject *kobj_lookup(struct kobj_map *, dev_t, int *);
 struct kobj_map *kobj_map_init(kobj_probe_t *, struct mutex *);
+void kobj_map_fini(struct kobj_map *);

 #endif
diff --git a/init/Kconfig b/init/Kconfig
index 0bb211a..5e1158e 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -292,6 +292,12 @@ config CONTAINER_DEBUG

    Say N if unsure

+config CONTAINER_DEVS
+ bool "Devices container subsystem"
+ depends on CONTAINERS
+ help
+   Controlls the visibility of devices
```

+
 config CONTAINER_NS
        bool "Namespace container subsystem"
        depends on CONTAINERS

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Cedric Le Goater on Mon, 24 Sep 2007 09:55:14 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Hi.
>
> At KS we have pointed out the need in some container, that allows
> to limit the visibility of some devices to task within it. I.e.
> allow for /dev/null, /dev/zero etc, but disable (by default) some
> IDE devices or SCSI discs and so on.
>
> Here's the beta of the container. Currently this only allows to
> hide the _character_ devices only from the living tasks. To play
> with it you just create the container like this
>
> # mount -t container none /cont/devs -o devices
> # mkdir /cont/devs/0
>
> it will have two specific files
>
> # ls /cont/devs
> devices.block devices.char notify_on_release releasable release_agent tasks
>
> then move a task into it
>
> # /bin/echo -n $$ > /cont/devs/0/tasks
>
> after this you won't be able to read from even /dev/zero
>
> # hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> meanwhile from another ssh session you will. You may allow access
> to /dev/zero like this
>
> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char

>
> More generally, the '+<major>:<minor>' string grants access to
> some device, and '-<major>:<minor>' disables one.
>
> The TODO list now looks like this:
> * add the block devices support :) don't know how to make it yet;

I think the mapping is done trough a pseudo-fs for the block devices.
It probably means that we will have to mount it multiple times to
handle the isolation.

> * make /proc/devices show relevant info depending on who is
>   reading it. currently even if major 1 is disabled for task,
>   it will be listed in this file;
> * make it possible to enable/disable not just individual major:minor
>   pair, but something more flexible, e.g. major:* for all minors
>   for given major or major:m1-m2 for minor range, etc;

yep.

> * add the ability to restrict the read/write permissions for a
>   container. currently one may just control the visible-invisible
>   state for a device in a container, but maybe just readable or
>   just writable would be better.
>
> This patch is minimally tested, because I just want to know your
> opinion on whether it worths developing the container in such a way or not.

it looks simple enough to me.

I'm wondering how many control groups subsystems we will need
to make The *Container* and if it's not worth just merging
them in a big unified one.

Thanks !

C.

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/drivers/base/map.c b/drivers/base/map.c
> index e87017f..0188053 100644
> --- a/drivers/base/map.c
> +++ b/drivers/base/map.c
> @@ -153,3 +153,21 @@ struct kobj_map *kobj_map_init(kobj_prob
>   p->lock = lock;

```
>   return p;
>  }
> +
> +void kobj_map_fini(struct kobj_map *map)
> +{
> + int i;
> + struct probe *p, *next;
> +
> + for (i = 0; i < 256; i++) {
> +  p = map->probes[i];
> +  while (p->next != NULL) {
> +   next = p->next;
> +   kfree(p);
> +   p = next;
> +  }
> + }
> +
> + kfree(p);
> + kfree(map);
> +}
> diff --git a/fs/Makefile b/fs/Makefile
> index 2661ef9..837c731 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -64,6 +64,8 @@ obj-y    += devpts/
>
>  obj-$(CONFIG_PROFILING)  += dcookies.o
>  obj-$(CONFIG_DLM)  += dlm/
> +
> +obj-$(CONFIG_CONTAINER_DEVS) += devscontrol.o
>
>  # Do not add any filesystems before this line
>  obj-$(CONFIG_REISERFS_FS) += reiserfs/
> diff --git a/fs/char_dev.c b/fs/char_dev.c
> index c3bfa76..1b0e4da 100644
> --- a/fs/char_dev.c
> +++ b/fs/char_dev.c
> @@ -22,6 +22,8 @@
>  #include <linux/mutex.h>
>  #include <linux/backing-dev.h>
>
> +#include <linux/devscontrol.h>
> +
>  #ifdef CONFIG_KMOD
>  #include <linux/kmod.h>
>  #endif
> @@ -362,17 +364,24 @@ int chrdev_open(struct inode * inode, st
>   struct cdev *p;
```

```
>   struct cdev *new = NULL;
>   int ret = 0;
> + struct kobj_map *map;
> +
> + map = task_cdev_map(current);
> + if (map == NULL)
> +  map = cdev_map;
>
>   spin_lock(&cdev_lock);
>   p = inode->i_cdev;
> - if (!p) {
> + if (!p || p->last != map) {
>    struct kobject *kobj;
>    int idx;
> +
>    spin_unlock(&cdev_lock);
> -  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
> +  kobj = kobj_lookup(map, inode->i_rdev, &idx);
>    if (!kobj)
>     return -ENXIO;
>    new = container_of(kobj, struct cdev, kobj);
> +  BUG_ON(p != NULL && p != new);
>    spin_lock(&cdev_lock);
>    p = inode->i_cdev;
>    if (!p) {
> @@ -384,6 +393,8 @@ int chrdev_open(struct inode * inode, st
>     ret = -ENXIO;
>   } else if (!cdev_get(p))
>    ret = -ENXIO;
> + if (p)
> +  p->last = map;
>   spin_unlock(&cdev_lock);
>   cdev_put(new);
>   if (ret)
> @@ -461,6 +472,49 @@ int cdev_add(struct cdev *p, dev_t dev,
>   return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
>  }
>
> +int cdev_add_to_map(struct kobj_map *map, dev_t dev)
> +{
> + int tmp;
> + struct kobject *k;
> + struct cdev *c;
> +
> + k = kobj_lookup(cdev_map, dev, &tmp);
> + if (k == NULL)
> +  return -ENODEV;
> +
```

```
> + c = container_of(k, struct cdev, kobj);
> + tmp = kobj_map(map, dev, 1, NULL, exact_match, exact_lock, c);
> + if (tmp < 0) {
> +  cdev_put(c);
> +  return tmp;
> + }
> +
> + return 0;
> +}
> +
> +int cdev_del_from_map(struct kobj_map *map, dev_t dev)
> +{
> + int tmp;
> + struct kobject *k;
> + struct cdev *c;
> +
> + k = kobj_lookup(cdev_map, dev, &tmp);
> + if (k == NULL)
> +  return -ENODEV;
> +
> + c = container_of(k, struct cdev, kobj);
> + kobj_unmap(map, dev, 1);
> +
> + spin_lock(&cdev_lock);
> + if (c->last == map)
> +  c->last = NULL;
> + spin_unlock(&cdev_lock);
> +
> + cdev_put(c);
> + cdev_put(c);
> + return 0;
> +}
> +
>  static void cdev_unmap(dev_t dev, unsigned count)
>  {
>   kobj_unmap(cdev_map, dev, count);
> @@ -542,6 +596,16 @@ static struct kobject *base_probe(dev_t
>   return NULL;
>  }
>
> +struct kobj_map *cdev_map_init(void)
> +{
> + return kobj_map_init(base_probe, &chrdevs_lock);
> +}
> +
> +void cdev_map_fini(struct kobj_map *map)
> +{
> + kobj_map_fini(map);
```

> +}
> +
> void __init chrdev_init(void)
> {
>   cdev_map = kobj_map_init(base_probe, &chrdevs_lock);
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..6fb5f05
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,170 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul@openvz.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/container.h>
> +#include <linux/cdev.h>
> +#include <linux/err.h>
> +#include <linux/devscontrol.h>
> +#include <linux/uaccess.h>
> +
> +struct devs_container {
> + struct container_subsys_state css;
> +
> + struct kobj_map *cdev_map;
> +};
> +
> +static inline
> +struct devs_container *css_to_devs(struct container_subsys_state *css)
> +{
> + return container_of(css, struct devs_container, css);
> +}
> +
> +static inline
> +struct devs_container *container_to_devs(struct container *cont)

```
> +{
> + return css_to_devs(container_subsys_state(cont, devs_subsys_id));
> +}
> +
> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + struct container_subsys_state *css;
> +
> + css = task_subsys_state(tsk, devs_subsys_id);
> + if (css->container->parent == NULL)
> +  return NULL;
> + else
> +  return css_to_devs(css)->cdev_map;
> +}
> +
> +static struct container_subsys_state *
> +devs_create(struct container_subsys *ss, struct container *cont)
> +{
> + struct devs_container *devs;
> +
> + devs = kzalloc(sizeof(struct devs_container), GFP_KERNEL);
> + if (devs == NULL)
> +  goto out;
> +
> + devs->cdev_map = cdev_map_init();
> + if (devs->cdev_map == NULL)
> +  goto out_free;
> +
> + return &devs->css;
> +
> +out_free:
> + kfree(devs);
> +out:
> + return ERR_PTR(-ENOMEM);
> +}
> +
> +static void devs_destroy(struct container_subsys *ss, struct container *cont)
> +{
> + struct devs_container *devs;
> +
> + devs = container_to_devs(cont);
> + cdev_map_fini(devs->cdev_map);
> + kfree(devs);
> +}
> +
> +static int decode_dev_name(char *buf, dev_t *dev)
> +{
> + unsigned int major, minor;
```

```
> + char *end;
> +
> + major = simple_strtoul(buf, &end, 10);
> + if (*end != ':')
> +  return -EINVAL;
> +
> + minor = simple_strtoul(end + 1, &end, 10);
> + if (*end != '\0')
> +  return -EINVAL;
> +
> + *dev = MKDEV(major, minor);
> + return 0;
> +}
> +
> +static ssize_t
> +devs_char_write(struct container *cont, struct cftype *cft, struct file *f,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + int err;
> + dev_t dev;
> + char buf[64];
> + struct devs_container *devs;
> +
> + if (copy_from_user(buf, userbuf, sizeof(buf)))
> +  return -EFAULT;
> +
> + err = decode_dev_name(buf + 1, &dev);
> + if (err < 0)
> +  return err;
> +
> + devs = container_to_devs(cont);
> +
> + switch (buf[0]) {
> + case '+':
> +  err = cdev_add_to_map(devs->cdev_map, dev);
> +  if (err < 0)
> +   return err;
> +
> +  css_get(&devs->css);
> +  break;
> + case '-':
> +  err = cdev_del_from_map(devs->cdev_map, dev);
> +  if (err < 0)
> +   return err;
> +
> +  css_put(&devs->css);
> +  break;
> + default:
```

```
> +  return -EINVAL;
> + }
> +
> + return nbytes;
> +}
> +
> +static ssize_t
> +devs_block_write(struct container *cont, struct cftype *cft, struct file *f,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + return -ENOTTY;
> +}
> +
> +static struct cftype devs_files[] = {
> + {
> +  .name = "char",
> +  .write = devs_char_write,
> + },
> + {
> +  .name = "block",
> +  .write = devs_block_write,
> + },
> +};
> +
> +static int devs_populate(struct container_subsys *ss, struct container *cont)
> +{
> + return container_add_files(cont, ss,
> +   devs_files, ARRAY_SIZE(devs_files));
> +}
> +
> +struct container_subsys devs_subsys = {
> + .name = "devices",
> + .subsys_id = devs_subsys_id,
> + .create = devs_create,
> + .destroy = devs_destroy,
> + .populate = devs_populate,
> +};
> diff --git a/include/linux/cdev.h b/include/linux/cdev.h
> index 1e29b13..0eded40 100644
> --- a/include/linux/cdev.h
> +++ b/include/linux/cdev.h
> @@ -9,6 +9,7 @@
>  struct file_operations;
>  struct inode;
>  struct module;
> +struct kobj_map;
>
>  struct cdev {
```

>   struct kobject kobj;
> @@ -17,6 +18,7 @@ struct cdev {
>   struct list_head list;
>   dev_t dev;
>   unsigned int count;
> + struct kobj_map *last;
> };
>
> void cdev_init(struct cdev *, const struct file_operations *);
> @@ -33,5 +35,9 @@ void cd_forget(struct inode *);
>
> extern struct backing_dev_info directly_mappable_cdev_bdi;
>
> +int cdev_add_to_map(struct kobj_map *map, dev_t dev);
> +int cdev_del_from_map(struct kobj_map *map, dev_t dev);
> +struct kobj_map *cdev_map_init(void);
> +void cdev_map_fini(struct kobj_map *map);
> #endif
> #endif
> diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
> index 81d11c2..9315a9b 100644
> --- a/include/linux/container_subsys.h
> +++ b/include/linux/container_subsys.h
> @@ -36,3 +36,9 @@ SUBSYS(mem_container)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CONTAINER_DEVS
> +SUBSYS(devs)
> +#endif
> +
> +/* */
> diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
> new file mode 100644
> index 0000000..51ae916
> --- /dev/null
> +++ b/include/linux/devscontrol.h
> @@ -0,0 +1,14 @@
> +#ifndef __DEVS_CONTROL_H__
> +#define __DEVS_CONTROL_H__
> +struct kobj_map;
> +struct task_struct;
> +
> +#ifdef CONFIG_CONTAINER_DEVS
> +struct kobj_map *task_cdev_map(struct task_struct *);
> +#else
> +static inline kobj_map *task_cdev_map(struct task_struct *tsk)

> +{
> + return NULL;
> +}
> +#endif
> +#endif
> diff --git a/include/linux/kobj_map.h b/include/linux/kobj_map.h
> index bafe178..2476f8d 100644
> --- a/include/linux/kobj_map.h
> +++ b/include/linux/kobj_map.h
> @@ -10,5 +10,6 @@ int kobj_map(struct kobj_map *, dev_t, u
> void kobj_unmap(struct kobj_map *, dev_t, unsigned long);
> struct kobject *kobj_lookup(struct kobj_map *, dev_t, int *);
> struct kobj_map *kobj_map_init(kobj_probe_t *, struct mutex *);
> +void kobj_map_fini(struct kobj_map *);
>
> #endif
> diff --git a/init/Kconfig b/init/Kconfig
> index 0bb211a..5e1158e 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -292,6 +292,12 @@ config CONTAINER_DEBUG
>
>   Say N if unsure
>
> +config CONTAINER_DEVS
> + bool "Devices container subsystem"
> + depends on CONTAINERS
> + help
> +   Controlls the visibility of devices
> +
> config CONTAINER_NS
>   bool "Namespace container subsystem"
>   depends on CONTAINERS
>

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Mon, 24 Sep 2007 11:47:26 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Pavel Emelyanov wrote:
>> Hi.

>>
>> At KS we have pointed out the need in some container, that allows
>> to limit the visibility of some devices to task within it. I.e.
>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>> IDE devices or SCSI discs and so on.
>>
>> Here's the beta of the container. Currently this only allows to
>> hide the _character_ devices only from the living tasks. To play
>> with it you just create the container like this
>>
>> # mount -t container none /cont/devs -o devices
>> # mkdir /cont/devs/0
>>
>> it will have two specific files
>>
>> # ls /cont/devs
>> devices.block devices.char notify_on_release releasable release_agent tasks
>>
>> then move a task into it
>>
>> # /bin/echo -n $$ > /cont/devs/0/tasks
>>
>> after this you won't be able to read from even /dev/zero
>>
>> # hexdump /dev/zero
>> hexdump: /dev/zero: No such device or address
>> hexdump: /dev/zero: Bad file descriptor
>>
>> meanwhile from another ssh session you will. You may allow access
>> to /dev/zero like this
>>
>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>
>> More generally, the '+<major>:<minor>' string grants access to
>> some device, and '-<major>:<minor>' disables one.
>>
>> The TODO list now looks like this:
>> * add the block devices support :) don't know how to make it yet;
>
> I think the mapping is done trough a pseudo-fs for the block devices.
> It probably means that we will have to mount it multiple times to
> handle the isolation.

Maybe. I looked over the block layer and found that character one
was simpler to start with.

>> * make /proc/devices show relevant info depending on who is
>>   reading it. currently even if major 1 is disabled for task,

>> it will be listed in this file;
>> * make it possible to enable/disable not just individual major:minor
>> pair, but something more flexible, e.g. major:* for all minors
>> for given major or major:m1-m2 for minor range, etc;
>
> yep.

:)

>> * add the ability to restrict the read/write permissions for a
>> container. currently one may just control the visible-invisible
>> state for a device in a container, but maybe just readable or
>> just writable would be better.
>>
>> This patch is minimally tested, because I just want to know your
>> opinion on whether it worths developing the container in such a way or not.
>
> it looks simple enough to me.

Well, OK. Then I will go on developing this one.

> I'm wondering how many control groups subsystems we will need
> to make The *Container* and if it's not worth just merging
> them in a big unified one.

Ha ha, so am I :)

> Thanks !
>
> C.

Thanks,
Pavel

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by serue on Mon, 24 Sep 2007 14:39:15 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Hi.
>
> At KS we have pointed out the need in some container, that allows
> to limit the visibility of some devices to task within it. I.e.

> allow for /dev/null, /dev/zero etc, but disable (by default) some
> IDE devices or SCSI discs and so on.
>
> Here's the beta of the container. Currently this only allows to
> hide the _character_ devices only from the living tasks. To play
> with it you just create the container like this
>
> # mount -t container none /cont/devs -o devices
> # mkdir /cont/devs/0
>
> it will have two specific files
>
> # ls /cont/devs
> devices.block devices.char notify_on_release releasable release_agent tasks
>
> then move a task into it
>
> # /bin/echo -n $$ > /cont/devs/0/tasks
>
> after this you won't be able to read from even /dev/zero
>
> # hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> meanwhile from another ssh session you will. You may allow access
> to /dev/zero like this
>
> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>
> More generally, the '+<major>:<minor>' string grants access to
> some device, and '-<major>:<minor>' disables one.
>
> The TODO list now looks like this:
> * add the block devices support :) don't know how to make it yet;
> * make /proc/devices show relevant info depending on who is
>   reading it. currently even if major 1 is disabled for task,
>   it will be listed in this file;
> * make it possible to enable/disable not just individual major:minor
>   pair, but something more flexible, e.g. major:* for all minors
>   for given major or major:m1-m2 for minor range, etc;
> * add the ability to restrict the read/write permissions for a
>   container. currently one may just control the visible-invisible
>   state for a device in a container, but maybe just readable or
>   just writable would be better.
>
> This patch is minimally tested, because I just want to know your
> opinion on whether it worths developing the container in such a way or not.

Hmm,

I was thinking we would use LSM for this. Mostly it should suffice
to set up a reasonable /dev for the container to start with, and
hook security_mknod() to prevent it creating devices not on it's
whitelist. If deemed necessary, read/write could be controlled
by hooking security_permission() and checking whether
file->f_path.dentry->d_inode is a device on the read or write
whitelist.

It would still be a device controller, so it can be composed with an
ns_proxy controller, and the whitelist is modified using the
devs_controller.whitelist file, but it registers a security_ops
with these two hooks.

I haven't implemented that yet, though, whereas you already have code :)
As for handling blkdevs with your code, would just hooking
fs/block_dev.c:do_open() not work? Or is that not what you are
asking?

thanks,
-serge

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/drivers/base/map.c b/drivers/base/map.c
> index e87017f..0188053 100644
> --- a/drivers/base/map.c
> +++ b/drivers/base/map.c
> @@ -153,3 +153,21 @@ struct kobj_map *kobj_map_init(kobj_prob
>   p->lock = lock;
>   return p;
> }
> +
> +void kobj_map_fini(struct kobj_map *map)
> +{
> + int i;
> + struct probe *p, *next;
> +
> + for (i = 0; i < 256; i++) {
> + p = map->probes[i];
> + while (p->next != NULL) {
> +   next = p->next;
> +   kfree(p);
> +   p = next;

```
> +  }
> + }
> +
> + kfree(p);
> + kfree(map);
> +}
> diff --git a/fs/Makefile b/fs/Makefile
> index 2661ef9..837c731 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -64,6 +64,8 @@ obj-y    += devpts/
>
>  obj-$(CONFIG_PROFILING)  += dcookies.o
>  obj-$(CONFIG_DLM)  += dlm/
> +
> +obj-$(CONFIG_CONTAINER_DEVS) += devscontrol.o
>
>  # Do not add any filesystems before this line
>  obj-$(CONFIG_REISERFS_FS) += reiserfs/
> diff --git a/fs/char_dev.c b/fs/char_dev.c
> index c3bfa76..1b0e4da 100644
> --- a/fs/char_dev.c
> +++ b/fs/char_dev.c
> @@ -22,6 +22,8 @@
>  #include <linux/mutex.h>
>  #include <linux/backing-dev.h>
>
> +#include <linux/devscontrol.h>
> +
>  #ifdef CONFIG_KMOD
>  #include <linux/kmod.h>
>  #endif
> @@ -362,17 +364,24 @@ int chrdev_open(struct inode * inode, st
>   struct cdev *p;
>   struct cdev *new = NULL;
>   int ret = 0;
> + struct kobj_map *map;
> +
> + map = task_cdev_map(current);
> + if (map == NULL)
> +  map = cdev_map;
>
>   spin_lock(&cdev_lock);
>   p = inode->i_cdev;
> - if (!p) {
> + if (!p || p->last != map) {
>    struct kobject *kobj;
>    int idx;
```

```
> +
>    spin_unlock(&cdev_lock);
> -  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
> +  kobj = kobj_lookup(map, inode->i_rdev, &idx);
>    if (!kobj)
>     return -ENXIO;
>    new = container_of(kobj, struct cdev, kobj);
> +  BUG_ON(p != NULL && p != new);
>    spin_lock(&cdev_lock);
>    p = inode->i_cdev;
>    if (!p) {
> @@ -384,6 +393,8 @@ int chrdev_open(struct inode * inode, st
>     ret = -ENXIO;
>   } else if (!cdev_get(p))
>    ret = -ENXIO;
> + if (p)
> +  p->last = map;
>   spin_unlock(&cdev_lock);
>   cdev_put(new);
>   if (ret)
> @@ -461,6 +472,49 @@ int cdev_add(struct cdev *p, dev_t dev,
>   return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
>  }
>
> +int cdev_add_to_map(struct kobj_map *map, dev_t dev)
> +{
> + int tmp;
> + struct kobject *k;
> + struct cdev *c;
> +
> + k = kobj_lookup(cdev_map, dev, &tmp);
> + if (k == NULL)
> +  return -ENODEV;
> +
> + c = container_of(k, struct cdev, kobj);
> + tmp = kobj_map(map, dev, 1, NULL, exact_match, exact_lock, c);
> + if (tmp < 0) {
> +  cdev_put(c);
> +  return tmp;
> + }
> +
> + return 0;
> +}
> +
> +int cdev_del_from_map(struct kobj_map *map, dev_t dev)
> +{
> + int tmp;
> + struct kobject *k;
```

```
> + struct cdev *c;
> +
> + k = kobj_lookup(cdev_map, dev, &tmp);
> + if (k == NULL)
> +  return -ENODEV;
> +
> + c = container_of(k, struct cdev, kobj);
> + kobj_unmap(map, dev, 1);
> +
> + spin_lock(&cdev_lock);
> + if (c->last == map)
> +  c->last = NULL;
> + spin_unlock(&cdev_lock);
> +
> + cdev_put(c);
> + cdev_put(c);
> + return 0;
> +}
> +
>  static void cdev_unmap(dev_t dev, unsigned count)
>  {
>   kobj_unmap(cdev_map, dev, count);
> @@ -542,6 +596,16 @@ static struct kobject *base_probe(dev_t
>   return NULL;
>  }
>
> +struct kobj_map *cdev_map_init(void)
> +{
> + return kobj_map_init(base_probe, &chrdevs_lock);
> +}
> +
> +void cdev_map_fini(struct kobj_map *map)
> +{
> + kobj_map_fini(map);
> +}
> +
>  void __init chrdev_init(void)
>  {
>   cdev_map = kobj_map_init(base_probe, &chrdevs_lock);
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..6fb5f05
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,170 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
```

```
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul@openvz.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/container.h>
> +#include <linux/cdev.h>
> +#include <linux/err.h>
> +#include <linux/devscontrol.h>
> +#include <linux/uaccess.h>
> +
> +struct devs_container {
> + struct container_subsys_state css;
> +
> + struct kobj_map *cdev_map;
> +};
> +
> +static inline
> +struct devs_container *css_to_devs(struct container_subsys_state *css)
> +{
> + return container_of(css, struct devs_container, css);
> +}
> +
> +static inline
> +struct devs_container *container_to_devs(struct container *cont)
> +{
> + return css_to_devs(container_subsys_state(cont, devs_subsys_id));
> +}
> +
> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + struct container_subsys_state *css;
> +
> + css = task_subsys_state(tsk, devs_subsys_id);
> + if (css->container->parent == NULL)
> +  return NULL;
> + else
> +  return css_to_devs(css)->cdev_map;
> +}
```

```
> +
> +static struct container_subsys_state *
> +devs_create(struct container_subsys *ss, struct container *cont)
> +{
> + struct devs_container *devs;
> +
> + devs = kzalloc(sizeof(struct devs_container), GFP_KERNEL);
> + if (devs == NULL)
> +  goto out;
> +
> + devs->cdev_map = cdev_map_init();
> + if (devs->cdev_map == NULL)
> +  goto out_free;
> +
> + return &devs->css;
> +
> +out_free:
> + kfree(devs);
> +out:
> + return ERR_PTR(-ENOMEM);
> +}
> +
> +static void devs_destroy(struct container_subsys *ss, struct container *cont)
> +{
> + struct devs_container *devs;
> +
> + devs = container_to_devs(cont);
> + cdev_map_fini(devs->cdev_map);
> + kfree(devs);
> +}
> +
> +static int decode_dev_name(char *buf, dev_t *dev)
> +{
> + unsigned int major, minor;
> + char *end;
> +
> + major = simple_strtoul(buf, &end, 10);
> + if (*end != ':')
> +  return -EINVAL;
> +
> + minor = simple_strtoul(end + 1, &end, 10);
> + if (*end != '\0')
> +  return -EINVAL;
> +
> + *dev = MKDEV(major, minor);
> + return 0;
> +}
> +
```

```
> +static ssize_t
> +devs_char_write(struct container *cont, struct cftype *cft, struct file *f,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + int err;
> + dev_t dev;
> + char buf[64];
> + struct devs_container *devs;
> +
> + if (copy_from_user(buf, userbuf, sizeof(buf)))
> +  return -EFAULT;
> +
> + err = decode_dev_name(buf + 1, &dev);
> + if (err < 0)
> +  return err;
> +
> + devs = container_to_devs(cont);
> +
> + switch (buf[0]) {
> + case '+':
> +  err = cdev_add_to_map(devs->cdev_map, dev);
> +  if (err < 0)
> +   return err;
> +
> +  css_get(&devs->css);
> +  break;
> + case '-':
> +  err = cdev_del_from_map(devs->cdev_map, dev);
> +  if (err < 0)
> +   return err;
> +
> +  css_put(&devs->css);
> +  break;
> + default:
> +  return -EINVAL;
> + }
> +
> + return nbytes;
> +}
> +
> +static ssize_t
> +devs_block_write(struct container *cont, struct cftype *cft, struct file *f,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + return -ENOTTY;
> +}
> +
> +static struct cftype devs_files[] = {
```

> + {
> + .name = "char",
> + .write = devs_char_write,
> + },
> + {
> + .name = "block",
> + .write = devs_block_write,
> + },
> +};
> +
> +static int devs_populate(struct container_subsys *ss, struct container *cont)
> +{
> + return container_add_files(cont, ss,
> +   devs_files, ARRAY_SIZE(devs_files));
> +}
> +
> +struct container_subsys devs_subsys = {
> + .name = "devices",
> + .subsys_id = devs_subsys_id,
> + .create = devs_create,
> + .destroy = devs_destroy,
> + .populate = devs_populate,
> +};
> diff --git a/include/linux/cdev.h b/include/linux/cdev.h
> index 1e29b13..0eded40 100644
> --- a/include/linux/cdev.h
> +++ b/include/linux/cdev.h
> @@ -9,6 +9,7 @@
> struct file_operations;
> struct inode;
> struct module;
> +struct kobj_map;
>
> struct cdev {
>   struct kobject kobj;
> @@ -17,6 +18,7 @@ struct cdev {
>   struct list_head list;
>   dev_t dev;
>   unsigned int count;
> + struct kobj_map *last;
> };
>
> void cdev_init(struct cdev *, const struct file_operations *);
> @@ -33,5 +35,9 @@ void cd_forget(struct inode *);
>
> extern struct backing_dev_info directly_mappable_cdev_bdi;
>
> +int cdev_add_to_map(struct kobj_map *map, dev_t dev);

> +int cdev_del_from_map(struct kobj_map *map, dev_t dev);
> +struct kobj_map *cdev_map_init(void);
> +void cdev_map_fini(struct kobj_map *map);
> #endif
> #endif
> diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
> index 81d11c2..9315a9b 100644
> --- a/include/linux/container_subsys.h
> +++ b/include/linux/container_subsys.h
> @@ -36,3 +36,9 @@ SUBSYS(mem_container)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CONTAINER_DEVS
> +SUBSYS(devs)
> +#endif
> +
> +/* */
> diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
> new file mode 100644
> index 0000000..51ae916
> --- /dev/null
> +++ b/include/linux/devscontrol.h
> @@ -0,0 +1,14 @@
> +#ifndef __DEVS_CONTROL_H__
> +#define __DEVS_CONTROL_H__
> +struct kobj_map;
> +struct task_struct;
> +
> +#ifdef CONFIG_CONTAINER_DEVS
> +struct kobj_map *task_cdev_map(struct task_struct *);
> +#else
> +static inline kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + return NULL;
> +}
> +#endif
> +#endif
> diff --git a/include/linux/kobj_map.h b/include/linux/kobj_map.h
> index bafe178..2476f8d 100644
> --- a/include/linux/kobj_map.h
> +++ b/include/linux/kobj_map.h
> @@ -10,5 +10,6 @@ int kobj_map(struct kobj_map *, dev_t, u
> void kobj_unmap(struct kobj_map *, dev_t, unsigned long);
> struct kobject *kobj_lookup(struct kobj_map *, dev_t, int *);
> struct kobj_map *kobj_map_init(kobj_probe_t *, struct mutex *);
> +void kobj_map_fini(struct kobj_map *);

>
> #endif
> diff --git a/init/Kconfig b/init/Kconfig
> index 0bb211a..5e1158e 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -292,6 +292,12 @@ config CONTAINER_DEBUG
>
>   Say N if unsure
>
> +config CONTAINER_DEVS
> + bool "Devices container subsystem"
> + depends on CONTAINERS
> + help
> +   Controlls the visibility of devices
> +
> config CONTAINER_NS
>   bool "Namespace container subsystem"
>   depends on CONTAINERS

_____

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Mon, 24 Sep 2007 14:58:10 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Hi.
>>
>> At KS we have pointed out the need in some container, that allows
>> to limit the visibility of some devices to task within it. I.e.
>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>> IDE devices or SCSI discs and so on.
>>
>> Here's the beta of the container. Currently this only allows to
>> hide the _character_ devices only from the living tasks. To play
>> with it you just create the container like this
>>
>> # mount -t container none /cont/devs -o devices
>> # mkdir /cont/devs/0
>>
>> it will have two specific files
>>
>> # ls /cont/devs

>> devices.block devices.char notify_on_release releasable release_agent tasks
>>
>> then move a task into it
>>
>> # /bin/echo -n $$ > /cont/devs/0/tasks
>>
>> after this you won't be able to read from even /dev/zero
>>
>> # hexdump /dev/zero
>> hexdump: /dev/zero: No such device or address
>> hexdump: /dev/zero: Bad file descriptor
>>
>> meanwhile from another ssh session you will. You may allow access
>> to /dev/zero like this
>>
>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>
>> More generally, the '+<major>:<minor>' string grants access to
>> some device, and '-<major>:<minor>' disables one.
>>
>> The TODO list now looks like this:
>> * add the block devices support :) don't know how to make it yet;
>> * make /proc/devices show relevant info depending on who is
>>   reading it. currently even if major 1 is disabled for task,
>>   it will be listed in this file;
>> * make it possible to enable/disable not just individual major:minor
>>   pair, but something more flexible, e.g. major:* for all minors
>>   for given major or major:m1-m2 for minor range, etc;
>> * add the ability to restrict the read/write permissions for a
>>   container. currently one may just control the visible-invisible
>>   state for a device in a container, but maybe just readable or
>>   just writable would be better.
>>
>> This patch is minimally tested, because I just want to know your
>> opinion on whether it worths developing the container in such a way or not.
>
> Hmm,
>
> I was thinking we would use LSM for this. Mostly it should suffice
> to set up a reasonable /dev for the container to start with, and
> hook security_mknod() to prevent it creating devices not on it's

Are you talking about disabling of mknod() for some files? No, please
no! This will break many... no - MANY tools inside such a container.

> whitelist. If deemed necessary, read/write could be controlled
> by hooking security_permission() and checking whether
> file->f_path.dentry->d_inode is a device on the read or write

> whitelist.
>
> It would still be a device controller, so it can be composed with an
> ns_proxy controller, and the whitelist is modified using the
> devs_controller.whitelist file, but it registers a security_ops
> with these two hooks.
>
> I haven't implemented that yet, though, whereas you already have code :)
> As for handling blkdevs with your code, would just hooking
> fs/block_dev.c:do_open() not work? Or is that not what you are
> asking?

Well, placing a hook into needed functions is something that can
work, of course, but this is not something that community would like
to see, so I tried to integrate them deeply.

> thanks,
> -serge

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by serue on Mon, 24 Sep 2007 15:20:01 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> Hi.
> >>
> >> At KS we have pointed out the need in some container, that allows
> >> to limit the visibility of some devices to task within it. I.e.
> >> allow for /dev/null, /dev/zero etc, but disable (by default) some
> >> IDE devices or SCSI discs and so on.
> >>
> >> Here's the beta of the container. Currently this only allows to
> >> hide the _character_ devices only from the living tasks. To play
> >> with it you just create the container like this
> >>
> >> # mount -t container none /cont/devs -o devices
> >> # mkdir /cont/devs/0
> >>
> >> it will have two specific files
> >>
> >> # ls /cont/devs

> >> devices.block devices.char notify_on_release releasable release_agent tasks
> >>
> >> then move a task into it
> >>
> >> # /bin/echo -n $$ > /cont/devs/0/tasks
> >>
> >> after this you won't be able to read from even /dev/zero
> >>
> >> # hexdump /dev/zero
> >> hexdump: /dev/zero: No such device or address
> >> hexdump: /dev/zero: Bad file descriptor
> >>
> >> meanwhile from another ssh session you will. You may allow access
> >> to /dev/zero like this
> >>
> >> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
> >>
> >> More generally, the '+<major>:<minor>' string grants access to
> >> some device, and '-<major>:<minor>' disables one.
> >>
> >> The TODO list now looks like this:
> >> * add the block devices support :) don't know how to make it yet;
> >> * make /proc/devices show relevant info depending on who is
> >>   reading it. currently even if major 1 is disabled for task,
> >>   it will be listed in this file;
> >> * make it possible to enable/disable not just individual major:minor
> >>   pair, but something more flexible, e.g. major:* for all minors
> >>   for given major or major:m1-m2 for minor range, etc;
> >> * add the ability to restrict the read/write permissions for a
> >>   container. currently one may just control the visible-invisible
> >>   state for a device in a container, but maybe just readable or
> >>   just writable would be better.
> >>
> >> This patch is minimally tested, because I just want to know your
> >> opinion on whether it worths developing the container in such a way or not.
> >
> > Hmm,
> >
> > I was thinking we would use LSM for this. Mostly it should suffice
> > to set up a reasonable /dev for the container to start with, and
> > hook security_mknod() to prevent it creating devices not on it's
>
> Are you talking about disabling of mknod() for some files? No, please
> no! This will break many... no - MANY tools inside such a container.

What's going to break if I don't allow mknod /dev/hda1? Is this during
standard /sbin/init for a container? And what does 'break' mean? If
you're not allowed to use the device, why should we pretend that you

can create it? Isn't that more devious?

A straight -EPERM on mknod just feels more warm+fuzzy to me. But if
things really are going to break to where you can't run a standard
distro in a container, then I guess we should go with your approach.

-serge

> > whitelist. If deemed necessary, read/write could be controlled
> > by hooking security_permission() and checking whether
> > file->f_path.dentry->d_inode is a device on the read or write
> > whitelist.
> >
> > It would still be a device controller, so it can be composed with an
> > ns_proxy controller, and the whitelist is modified using the
> > devs_controller.whitelist file, but it registers a security_ops
> > with these two hooks.
> >
> > I haven't implemented that yet, though, whereas you already have code :)
> > As for handling blkdevs with your code, would just hooking
> > fs/block_dev.c:do_open() not work? Or is that not what you are
> > asking?
>
> Well, placing a hook into needed functions is something that can
> work, of course, but this is not something that community would like
> to see, so I tried to integrate them deeply.
>
> > thanks,
> > -serge

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by dev on Mon, 24 Sep 2007 15:31:19 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>
>>Serge E. Hallyn wrote:
>>
>>>Quoting Pavel Emelyanov (xemul@openvz.org):
>>>
>>>>Hi.
>>>>

>>>>At KS we have pointed out the need in some container, that allows
>>>>to limit the visibility of some devices to task within it. I.e.
>>>>allow for /dev/null, /dev/zero etc, but disable (by default) some
>>>>IDE devices or SCSI discs and so on.
>>>>
>>>>Here's the beta of the container. Currently this only allows to
>>>>hide the _character_ devices only from the living tasks. To play
>>>>with it you just create the container like this
>>>>
>>>> # mount -t container none /cont/devs -o devices
>>>> # mkdir /cont/devs/0
>>>>
>>>>it will have two specific files
>>>>
>>>> # ls /cont/devs
>>>>devices.block devices.char notify_on_release releasable release_agent tasks
>>>>
>>>>then move a task into it
>>>>
>>>> # /bin/echo -n $$ > /cont/devs/0/tasks
>>>>
>>>>after this you won't be able to read from even /dev/zero
>>>>
>>>> # hexdump /dev/zero
>>>>hexdump: /dev/zero: No such device or address
>>>>hexdump: /dev/zero: Bad file descriptor
>>>>
>>>>meanwhile from another ssh session you will. You may allow access
>>>>to /dev/zero like this
>>>>
>>>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>>>
>>>>More generally, the '+<major>:<minor>' string grants access to
>>>>some device, and '-<major>:<minor>' disables one.
>>>>
>>>>The TODO list now looks like this:
>>>>* add the block devices support :) don't know how to make it yet;
>>>>* make /proc/devices show relevant info depending on who is
>>>> reading it. currently even if major 1 is disabled for task,
>>>> it will be listed in this file;
>>>>* make it possible to enable/disable not just individual major:minor
>>>> pair, but something more flexible, e.g. major:* for all minors
>>>> for given major or major:m1-m2 for minor range, etc;
>>>>* add the ability to restrict the read/write permissions for a
>>>> container. currently one may just control the visible-invisible
>>>> state for a device in a container, but maybe just readable or
>>>> just writable would be better.
>>>>

>>>>This patch is minimally tested, because I just want to know your
>>>>opinion on whether it worths developing the container in such a way or not.
>>>
>>>Hmm,
>>>
>>>I was thinking we would use LSM for this. Mostly it should suffice
>>>to set up a reasonable /dev for the container to start with, and
>>>hook security_mknod() to prevent it creating devices not on it's
>>
>>Are you talking about disabling of mknod() for some files? No, please
>>no! This will break many... no - MANY tools inside such a container.
>
>
> What's going to break if I don't allow mknod /dev/hda1? Is this during
> standard /sbin/init for a container? And what does 'break' mean? If
> you're not allowed to use the device, why should we pretend that you
> can create it? Isn't that more devious?
>
> A straight -EPERM on mknod just feels more warm+fuzzy to me. But if
> things really are going to break to where you can't run a standard
> distro in a container, then I guess we should go with your approach.

at least:
- udev which dynamically creates dev nodes including static devices.
- device nodes in RPM's. rpm installation should not fail.

I remember there were others, but in general mknod from root should not fail
until there is ENOSPC. And EPERM is handled by applications on open much better
then on creation, since applications are ready that they are executed errorneously
under wrong user account.

Thanks,
Kirill

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Mon, 24 Sep 2007 15:31:57 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelyanov (xemul@openvz.org):

>>>> Hi.
>>>>
>>>> At KS we have pointed out the need in some container, that allows
>>>> to limit the visibility of some devices to task within it. I.e.
>>>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>>>> IDE devices or SCSI discs and so on.
>>>>
>>>> Here's the beta of the container. Currently this only allows to
>>>> hide the _character_ devices only from the living tasks. To play
>>>> with it you just create the container like this
>>>>
>>>> # mount -t container none /cont/devs -o devices
>>>> # mkdir /cont/devs/0
>>>>
>>>> it will have two specific files
>>>>
>>>> # ls /cont/devs
>>>> devices.block devices.char notify_on_release releasable release_agent tasks
>>>>
>>>> then move a task into it
>>>>
>>>> # /bin/echo -n $$ > /cont/devs/0/tasks
>>>>
>>>> after this you won't be able to read from even /dev/zero
>>>>
>>>> # hexdump /dev/zero
>>>> hexdump: /dev/zero: No such device or address
>>>> hexdump: /dev/zero: Bad file descriptor
>>>>
>>>> meanwhile from another ssh session you will. You may allow access
>>>> to /dev/zero like this
>>>>
>>>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>>>
>>>> More generally, the '+<major>:<minor>' string grants access to
>>>> some device, and '-<major>:<minor>' disables one.
>>>>
>>>> The TODO list now looks like this:
>>>> * add the block devices support :) don't know how to make it yet;
>>>> * make /proc/devices show relevant info depending on who is
>>>>   reading it. currently even if major 1 is disabled for task,
>>>>   it will be listed in this file;
>>>> * make it possible to enable/disable not just individual major:minor
>>>>   pair, but something more flexible, e.g. major:* for all minors
>>>>   for given major or major:m1-m2 for minor range, etc;
>>>> * add the ability to restrict the read/write permissions for a
>>>>   container. currently one may just control the visible-invisible
>>>>   state for a device in a container, but maybe just readable or

>>>> just writable would be better.
>>>>
>>>> This patch is minimally tested, because I just want to know your
>>>> opinion on whether it worths developing the container in such a way or not.
>>> Hmm,
>>>
>>> I was thinking we would use LSM for this. Mostly it should suffice
>>> to set up a reasonable /dev for the container to start with, and
>>> hook security_mknod() to prevent it creating devices not on it's
>> Are you talking about disabling of mknod() for some files? No, please
>> no! This will break many... no - MANY tools inside such a container.
>
> What's going to break if I don't allow mknod /dev/hda1? Is this during
> standard /sbin/init for a container? And what does 'break' mean? If
> you're not allowed to use the device, why should we pretend that you
> can create it? Isn't that more devious?

Standard linux kernel allows you to create any devices you wish,
so container must operate the same way.

Besides, what to do if you have enables some device to it, then the
container user creates it and after this you disable it again. In this
case user will still be able to open the device and work with it :(
With my approach we will return -EPERM during this open :)

Or some better example - container owner mounts some external ext3
partitions with plenty of deices on it. No way to disable their
usage unless you control their open().

> A straight -EPERM on mknod just feels more warm+fuzzy to me. But if
> things really are going to break to where you can't run a standard
> distro in a container, then I guess we should go with your approach.

If udef fails to create a statically requested device it may break.
With broken udev no containers will work (using some latest distros).

Moreover - if you later grant access to this device udev won't try
to re-create it again unless specially asked.

> -serge

Thanks,
Pavel

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by serue on Mon, 24 Sep 2007 15:51:39 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> Serge E. Hallyn wrote:
> >>> Quoting Pavel Emelyanov (xemul@openvz.org):
> >>>> Hi.
> >>>>
> >>>> At KS we have pointed out the need in some container, that allows
> >>>> to limit the visibility of some devices to task within it. I.e.
> >>>> allow for /dev/null, /dev/zero etc, but disable (by default) some
> >>>> IDE devices or SCSI discs and so on.
> >>>>
> >>>> Here's the beta of the container. Currently this only allows to
> >>>> hide the _character_ devices only from the living tasks. To play
> >>>> with it you just create the container like this
> >>>>
> >>>> # mount -t container none /cont/devs -o devices
> >>>> # mkdir /cont/devs/0
> >>>>
> >>>> it will have two specific files
> >>>>
> >>>> # ls /cont/devs
> >>>> devices.block devices.char notify_on_release releasable release_agent tasks
> >>>>
> >>>> then move a task into it
> >>>>
> >>>> # /bin/echo -n $$ > /cont/devs/0/tasks
> >>>>
> >>>> after this you won't be able to read from even /dev/zero
> >>>>
> >>>> # hexdump /dev/zero
> >>>> hexdump: /dev/zero: No such device or address
> >>>> hexdump: /dev/zero: Bad file descriptor
> >>>>
> >>>> meanwhile from another ssh session you will. You may allow access
> >>>> to /dev/zero like this
> >>>>
> >>>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
> >>>>
> >>>> More generally, the '+<major>:<minor>' string grants access to
> >>>> some device, and '-<major>:<minor>' disables one.
> >>>>
> >>>> The TODO list now looks like this:
> >>>> * add the block devices support :) don't know how to make it yet;
> >>>> * make /proc/devices show relevant info depending on who is

> >>>> reading it. currently even if major 1 is disabled for task,
> >>>> it will be listed in this file;
> >>>> * make it possible to enable/disable not just individual major:minor
> >>>> pair, but something more flexible, e.g. major:* for all minors
> >>>> for given major or major:m1-m2 for minor range, etc;
> >>>> * add the ability to restrict the read/write permissions for a
> >>>> container. currently one may just control the visible-invisible
> >>>> state for a device in a container, but maybe just readable or
> >>>> just writable would be better.
> >>>>
> >>>> This patch is minimally tested, because I just want to know your
> >>>> opinion on whether it worths developing the container in such a way or not.
> >>> Hmm,
> >>>
> >>> I was thinking we would use LSM for this. Mostly it should suffice
> >>> to set up a reasonable /dev for the container to start with, and
> >>> hook security_mknod() to prevent it creating devices not on it's
> >> Are you talking about disabling of mknod() for some files? No, please
> >> no! This will break many... no - MANY tools inside such a container.
> >
> > What's going to break if I don't allow mknod /dev/hda1? Is this during
> > standard /sbin/init for a container? And what does 'break' mean? If
> > you're not allowed to use the device, why should we pretend that you
> > can create it? Isn't that more devious?
>
> Standard linux kernel allows you to create any devices you wish,
> so container must operate the same way.

And security_mknod() in standard linux kernel allows you to prevent it
being created.

> Besides, what to do if you have enables some device to it, then the
> container user creates it and after this you disable it again. In this

Well then you need to hire a new admin who can't be bribed :)

But seriously, that's where security_file_permission() would have a
check on open.

> case user will still be able to open the device and work with it :(
> With my approach we will return -EPERM during this open :)

So would security_file_permission().

> Or some better example - container owner mounts some external ext3
> partitions with plenty of deices on it. No way to disable their
> usage unless you control their open().

We can control their open(), control their read/write, and and make
their external ext3 be mounted NODEV to boot if we want.

> > A straight -EPERM on mknod just feels more warm+fuzzy to me. But if
> > things really are going to break to where you can't run a standard
> > distro in a container, then I guess we should go with your approach.
>
> If udef fails to create a statically requested device it may break.
> With broken udev no containers will work (using some latest distros).

This sounds like both the best and the worst argument all wrapped into
one :)

If udev is going to fail, and the container won't 'boot', and there is
no way around this, then maybe my approach isn't workable.

On the other hand, if udev isn't allowed to create some device, it
should just fail and move on. No excuse for it to fail the whole
system boot.

> Moreover - if you later grant access to this device udev won't try
> to re-create it again unless specially asked.

So?

Now again, mind you I'm just offering an alternative. I'm not objecting
to your patch, other than that it seems less straightforward to me.

Maybe it's best if i just code up an example. I'll be gone part of the
week, and should send out new versions of some other patches first, but
getting two approaches to this to compare and contrast can't hurt.

thanks,
-serge

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Cedric Le Goater on Mon, 24 Sep 2007 15:57:12 GMT
View Forum Message <> Reply to Message

[...]

> Maybe it's best if i just code up an example. I'll be gone part of the
> week, and should send out new versions of some other patches first, but

> getting two approaches to this to compare and contrast can't hurt.

I guess we will need a control group subsystem in both approach, right ?

C.

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by serue on Mon, 24 Sep 2007 16:18:23 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> [...]
>
> > Maybe it's best if i just code up an example. I'll be gone part of the
> > week, and should send out new versions of some other patches first, but
> > getting two approaches to this to compare and contrast can't hurt.
>
> I guess we will need a control group subsystem in both approach, right ?

Yes. Pavel's has one, and I would use one both for task tracking and
to use the cgroup vfs interface to construct the device whitelists.

thanks,
-serge

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by ebiederm on Mon, 24 Sep 2007 16:47:50 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov <xemul@openvz.org> writes:

> Hi.
>
> At KS we have pointed out the need in some container, that allows
> to limit the visibility of some devices to task within it. I.e.
> allow for /dev/null, /dev/zero etc, but disable (by default) some
> IDE devices or SCSI discs and so on.

NAK

We do not want a control group subsystem for this.

For the short term we can just drop CAP_SYS_MKNOD.

For the long term we need a device namespace for application
migration, so they can continue to use devices with the same
major+minor number pair after the migration event. Things like
ensuring a call to stat on a given file before and after the migration
return the exact same information sounds compelling. So I don't think
this is even strictly limited to virtual devices anymore. How many
applications are there out there that memorize the stat data on a file
and so they can detect if it has changed?

If we need something between those two it may make sense to enhance
the LSM or perhaps introduce an alternate set security hooks. Still
if we are going to need a full device namespace that may be a little
much.

Eric

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by serue on Mon, 24 Sep 2007 16:53:07 GMT
View Forum Message <> Reply to Message

Quoting Kirill Korotaev (dev@sw.ru):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >
> >>Serge E. Hallyn wrote:
> >>
> >>>Quoting Pavel Emelyanov (xemul@openvz.org):
> >>>
> >>>>Hi.
> >>>>
> >>>>At KS we have pointed out the need in some container, that allows
> >>>>to limit the visibility of some devices to task within it. I.e.
> >>>>allow for /dev/null, /dev/zero etc, but disable (by default) some
> >>>>IDE devices or SCSI discs and so on.
> >>>>
> >>>>Here's the beta of the container. Currently this only allows to

> >>>>hide the _character_ devices only from the living tasks. To play
> >>>>with it you just create the container like this
> >>>>
> >>>> # mount -t container none /cont/devs -o devices
> >>>> # mkdir /cont/devs/0
> >>>>
> >>>>it will have two specific files
> >>>>
> >>>> # ls /cont/devs
> >>>>devices.block devices.char notify_on_release releasable release_agent tasks
> >>>>
> >>>>then move a task into it
> >>>>
> >>>> # /bin/echo -n $$ > /cont/devs/0/tasks
> >>>>
> >>>>after this you won't be able to read from even /dev/zero
> >>>>
> >>>> # hexdump /dev/zero
> >>>>hexdump: /dev/zero: No such device or address
> >>>>hexdump: /dev/zero: Bad file descriptor
> >>>>
> >>>>meanwhile from another ssh session you will. You may allow access
> >>>>to /dev/zero like this
> >>>>
> >>>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
> >>>>
> >>>>More generally, the '+<major>:<minor>' string grants access to
> >>>>some device, and '-<major>:<minor>' disables one.
> >>>>
> >>>>The TODO list now looks like this:
> >>>>* add the block devices support :) don't know how to make it yet;
> >>>>* make /proc/devices show relevant info depending on who is
> >>>> reading it. currently even if major 1 is disabled for task,
> >>>> it will be listed in this file;
> >>>>* make it possible to enable/disable not just individual major:minor
> >>>> pair, but something more flexible, e.g. major:* for all minors
> >>>> for given major or major:m1-m2 for minor range, etc;
> >>>>* add the ability to restrict the read/write permissions for a
> >>>> container. currently one may just control the visible-invisible
> >>>> state for a device in a container, but maybe just readable or
> >>>> just writable would be better.
> >>>>
> >>>>This patch is minimally tested, because I just want to know your
> >>>>opinion on whether it worths developing the container in such a way or not.
> >>>
> >>>Hmm,
> >>>
> >>>I was thinking we would use LSM for this. Mostly it should suffice

> >>>to set up a reasonable /dev for the container to start with, and
> >>>hook security_mknod() to prevent it creating devices not on it's
> >>
> >>Are you talking about disabling of mknod() for some files? No, please
> >>no! This will break many... no - MANY tools inside such a container.
> >
> >
> > What's going to break if I don't allow mknod /dev/hda1? Is this during
> > standard /sbin/init for a container? And what does 'break' mean? If
> > you're not allowed to use the device, why should we pretend that you
> > can create it? Isn't that more devious?
> >
> > A straight -EPERM on mknod just feels more warm+fuzzy to me. But if
> > things really are going to break to where you can't run a standard
> > distro in a container, then I guess we should go with your approach.
>
> at least:
> - udev which dynamically creates dev nodes including static devices.
> - device nodes in RPM's. rpm installation should not fail.
>
> I remember there were others, but in general mknod from root should not fail
> until there is ENOSPC. And EPERM is handled by applications on open much better
> then on creation, since applications are ready that they are executed errorneously
> under wrong user account.

We'll need a way to prevent collusion. For instance uid 1000 on the
system starts a new container where he is root. There he creates a node
hda1 someplace and allows uid 1000 in the host container to read/write
it... Certain for normal files we want to allow such sharing.

> Thanks,
> Kirill

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Tue, 25 Sep 2007 07:48:03 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelyanov <xemul@openvz.org> writes:
>
>> Hi.
>>
>> At KS we have pointed out the need in some container, that allows

>> to limit the visibility of some devices to task within it. I.e.
>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>> IDE devices or SCSI discs and so on.
>
> NAK
>
> We do not want a control group subsystem for this.
>
> For the short term we can just drop CAP_SYS_MKNOD.
>
> For the long term we need a device namespace for application
> migration, so they can continue to use devices with the same
> major+minor number pair after the migration event. Things like

Oh! Can you provide us an example when after the migration some
device's major+minor pair change on the same device?

> ensuring a call to stat on a given file before and after the migration
> return the exact same information sounds compelling. So I don't think
> this is even strictly limited to virtual devices anymore. How many
> applications are there out there that memorize the stat data on a file
> and so they can detect if it has changed?
>
> If we need something between those two it may make sense to enhance
> the LSM or perhaps introduce an alternate set security hooks. Still
> if we are going to need a full device namespace that may be a little
> much.
>
> Eric
>

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Paul Menage on Tue, 25 Sep 2007 07:53:11 GMT
View Forum Message <> Reply to Message

On 9/24/07, Pavel Emelyanov <xemul@openvz.org> wrote:
>
> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>
> More generally, the '+<major>:<minor>' string grants access to
> some device, and '-<major>:<minor>' disables one.

How about a more forward-compatible API:

<major>:<minor>=<permissions>[,<remapped_major>:<remapped_minor>]

This would allow you the control the access that each cgroup has to a
given device (permissions of 0 indicates that the device isn't even
visible, i.e. the same as your "-<major>:<minor>" operation. For now
specifying the (optional) remapping could just fail, but at least the
API would be defined.

Paul

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Tue, 25 Sep 2007 08:00:07 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 9/24/07, Pavel Emelyanov <xemul@openvz.org> wrote:
>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>
>> More generally, the '+<major>:<minor>' string grants access to
>> some device, and '-<major>:<minor>' disables one.
>
> How about a more forward-compatible API:
>
> <major>:<minor>=<permissions>[,<remapped_major>:<remapped_minor>]

I'd rather make it look like

<major>:<mino>[:<permissions>][:<map_major>:<map_minor>]

where

<permissions>:=[r-][w-] and NULL means rw

this would keep current API compatible and allow it for extension.

> This would allow you the control the access that each cgroup has to a
> given device (permissions of 0 indicates that the device isn't even
> visible, i.e. the same as your "-<major>:<minor>" operation. For now
> specifying the (optional) remapping could just fail, but at least the
> API would be defined.
>

> Paul
>

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Cedric Le Goater on Tue, 25 Sep 2007 11:20:31 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Paul Menage wrote:
>> On 9/24/07, Pavel Emelyanov <xemul@openvz.org> wrote:
>>> # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
>>>
>>> More generally, the '+<major>:<minor>' string grants access to
>>> some device, and '-<major>:<minor>' disables one.
>> How about a more forward-compatible API:
>>
>> <major>:<minor>=<permissions>[,<remapped_major>:<remapped_minor>]
>
> I'd rather make it look like
>
> <major>:<mino>[:<permissions>][:<map_major>:<map_minor>]
>
> where
>
> <permissions>:=[r-][w-] and NULL means rw

bah. numeric mode are better.

C.

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Cedric Le Goater on Tue, 25 Sep 2007 12:25:24 GMT
View Forum Message <> Reply to Message

Hello Eric !

Eric W. Biederman wrote:
> Pavel Emelyanov <xemul@openvz.org> writes:
>
>> At KS we have pointed out the need in some container, that allows
>> to limit the visibility of some devices to task within it. I.e.
>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>> IDE devices or SCSI discs and so on.
>
> NAK
>
> We do not want a control group subsystem for this.

we will need one way to configure the list of available devices from
user space. Any proposal ?

> For the short term we can just drop CAP_SYS_MKNOD.

Sure. Pavel is working on something mid-term ;)

> For the long term we need a device namespace for application
> migration, so they can continue to use devices with the same
> major+minor number pair after the migration event.

Hmm, yes. I can imagine that for some big database application using
raw devices but it only means that the same device must be present
upon restart. I don't see any identifier virtualization issues.

> Things like
> ensuring a call to stat on a given file before and after the migration
> return the exact same information sounds compelling. So I don't think
> this is even strictly limited to virtual devices anymore. How many
> applications are there out there that memorize the stat data on a file
> and so they can detect if it has changed?

that we need to support of course, otherwise we would break things
like tail.

> If we need something between those two it may make sense to enhance
> the LSM or perhaps introduce an alternate set security hooks. Still
> if we are going to need a full device namespace that may be a little
> much.

serge's implementation using security hooks should help us choose
the right approach.

Thanks !

C.

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

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by ebiederm on Tue, 25 Sep 2007 13:30:44 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov <xemul@openvz.org> writes:
>
> Oh! Can you provide us an example when after the migration some
> device's major+minor pair change on the same device?

SCSI disks on a SAN. Network accessible block devices.
All kinds of logical/virtual devices like ttys, the loop device, and
ramdisks.

It isn't especially frequent that something cares, but fundamentally
the same issues apply.

Eric

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

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by ebiederm on Tue, 25 Sep 2007 13:43:53 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Hello Eric !
>
> Eric W. Biederman wrote:
>> Pavel Emelyanov <xemul@openvz.org> writes:
>>
>>> At KS we have pointed out the need in some container, that allows
>>> to limit the visibility of some devices to task within it. I.e.
>>> allow for /dev/null, /dev/zero etc, but disable (by default) some
>>> IDE devices or SCSI discs and so on.
>>
>> NAK

>>
>> We do not want a control group subsystem for this.
>
> we will need one way to configure the list of available devices from
> user space. Any proposal ?

Proposal 1/2. From the kernel side we have.
dev_ns_add(kdev_t cur_dev, struct dev_ns *target_ns, kdev_t target_kdev)
Which looks up the device and add it to the hash tables in the proper
device namespace, and fires off the appropriate hotplug events.

I guess the easy user space interface would be:
echo <device_ns_pid>:<major>:<minor> > /sys/block/ram0/dev

Although I suspect that we want some restrictions on what
combinations of major and minor numbers are valid.

Despite the fact that my gut says writeable sysfs files were
a bad idea. Since we have them my gut says sysfs the filesystem
of devices is where we need the control files for devices.

>> For the short term we can just drop CAP_SYS_MKNOD.
>
> Sure. Pavel is working on something mid-term ;)

Well. I don't think midterm is mergeable, I do think it is good
for conversation though. I also don't see why what Pavel is doing
can't be implemented as a device namespace.

>> For the long term we need a device namespace for application
>> migration, so they can continue to use devices with the same
>> major+minor number pair after the migration event.
>
> Hmm, yes. I can imagine that for some big database application using
> raw devices but it only means that the same device must be present
> upon restart. I don't see any identifier virtualization issues.

Well there is the classic one. You are migrating to a machine which
is using that major+minor number for a different device already.

Especially in the cases like network block devices or SCSI talking
to SAN, we can talk to the same device and still have a different
major+minor number after migration in the current setup.

I think we can hit similar issues with ttys, loopback devices,
and ramdisks as well.

>> Things like

>> ensuring a call to stat on a given file before and after the migration
>> return the exact same information sounds compelling. So I don't think
>> this is even strictly limited to virtual devices anymore. How many
>> applications are there out there that memorize the stat data on a file
>> and so they can detect if it has changed?
>
> that we need to support of course, otherwise we would break things
> like tail.

Exactly. tail, git, backup software.
All kinds of infrequently run but interesting software.

>> If we need something between those two it may make sense to enhance
>> the LSM or perhaps introduce an alternate set security hooks. Still
>> if we are going to need a full device namespace that may be a little
>> much.
>
> serge's implementation using security hooks should help us choose
> the right approach.

Sure.

Currently I have to agree with Alan Cox that our biggest security
need seems to be a good implementation of revoke in the kernel.
So we can do things like ensure a device is not being used by anyone
else. For removal of character and block devices we may not need a
general thing but it is worth looking at.

Eric

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Dave Hansen on Wed, 26 Sep 2007 15:36:12 GMT
View Forum Message <> Reply to Message

On Tue, 2007-09-25 at 07:30 -0600, Eric W. Biederman wrote:
> Pavel Emelyanov <xemul@openvz.org> writes:
> >
> > Oh! Can you provide us an example when after the migration some
> > device's major+minor pair change on the same device?
>
> SCSI disks on a SAN. Network accessible block devices.
> All kinds of logical/virtual devices like ttys, the loop device, and
> ramdisks.

>
> It isn't especially frequent that something cares, but fundamentally
> the same issues apply.

To be clear, this just covers cases where an application has
internalized the device number, right?

Most applications should be pretty happy with the devices having
persistent device names across a restart, and we can do that with udev
and no kernel patching.

-- Dave

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by ebiederm on Wed, 26 Sep 2007 19:09:46 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Tue, 2007-09-25 at 07:30 -0600, Eric W. Biederman wrote:
>> Pavel Emelyanov <xemul@openvz.org> writes:
>> >
>> > Oh! Can you provide us an example when after the migration some
>> > device's major+minor pair change on the same device?
>>
>> SCSI disks on a SAN. Network accessible block devices.
>> All kinds of logical/virtual devices like ttys, the loop device, and
>> ramdisks.
>>
>> It isn't especially frequent that something cares, but fundamentally
>> the same issues apply.
>
> To be clear, this just covers cases where an application has
> _internalized_ the device number, right?

Also cases where you want to call mknod in the container.

> Most applications should be pretty happy with the devices having
> persistent device names across a restart, and we can do that with udev
> and no kernel patching.

Yes. But the applications that do internalize stat data from files

aren't that uncommon. git, and backup software etc.

There is also a fair bit of work that is needed to get sysfs
and the hotplug events isolated, when we start allowing mknod etc.

Basically if I figure if we are going to deal with this we need to handle
the entire problem because these pieces are user visible. I don't
think it is a great priority.

Eric

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

---

Subject: Re: [RFC][PATCH] Devices visibility container
Posted by Dave Hansen on Thu, 27 Sep 2007 15:46:45 GMT
View Forum Message <> Reply to Message

On Wed, 2007-09-26 at 13:09 -0600, Eric W. Biederman wrote:
> Dave Hansen <haveblue@us.ibm.com> writes:
>
> > On Tue, 2007-09-25 at 07:30 -0600, Eric W. Biederman wrote:
> >> Pavel Emelyanov <xemul@openvz.org> writes:
> >> >
> >> > Oh! Can you provide us an example when after the migration some
> >> > device's major+minor pair change on the same device?
> >>
> >> SCSI disks on a SAN. Network accessible block devices.
> >> All kinds of logical/virtual devices like ttys, the loop device, and
> >> ramdisks.
> >>
> >> It isn't especially frequent that something cares, but fundamentally
> >> the same issues apply.
> >
> > To be clear, this just covers cases where an application has
> > _internalized_ the device number, right?
>
> Also cases where you want to call mknod in the container.

mknod of device files only, yeah.

> > Most applications should be pretty happy with the devices having
> > persistent device names across a restart, and we can do that with udev
> > and no kernel patching.
>
> Yes. But the applications that do internalize stat data from files

> aren't that uncommon. git, and backup software etc.
>
> There is also a fair bit of work that is needed to get sysfs
> and the hotplug events isolated, when we start allowing mknod etc.
>
> Basically if I figure if we are going to deal with this we need to handle
> the entire problem because these pieces are user visible. I don't
> think it is a great priority.

Exactly. We have to allow mknod before any of this gets interesting in
the least.

-- Dave

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