
Subject: [PATCH] cgroups: implement device whitelist lsm (v3)
Posted by [serue](#) on Thu, 13 Mar 2008 14:41:42 GMT

[View Forum Message](#) <> [Reply to Message](#)

Implement a cgroup using the LSM interface to enforce open and mknod on device files.

This implements a simple device access whitelist. A whitelist entry has 4 fields. 'type' is a (all), c (char), or b (block). 'all' means it applies to all types, all major numbers, and all minor numbers. Major and minor are obvious. Access is a composition of r (read), w (write), and m (mknod).

The root devcgroup starts with rwm to 'all'. A child devcg gets a copy of the parent. Admins can then add and remove devices to the whitelist. Once CAP_HOST_ADMIN is introduced it will be needed to add entries as well or remove entries from another cgroup, though just CAP_SYS_ADMIN will suffice to remove entries for your own group.

An entry is added by doing "echo <type> <maj> <min> <access>" > devcg.allow, for instance:

```
echo b 7 0 mrw > /cgroups/1/devcg.allow
```

An entry is removed by doing likewise into devcg.deny. Since this is a pure whitelist, not acls, you can only remove entries which exist in the whitelist. You must explicitly

```
echo a 0 0 mrw > /cgroups/1/devcg.deny
```

to remove the "allow all" entry which is automatically inherited from the root cgroup.

While composing this with the ns_cgroup may seem logical, it is not the right thing to do, because updates to /cg/cg1/devcg.deny are not reflected in /cg/cg1/cg2/devcg.allow.

A task may only be moved to another devcgroup if it is moving to a direct descendent of its current devcgroup.

CAP_NS_OVERRIDE is defined as the capability needed to cross namespaces. A task needs both CAP_NS_OVERRIDE and CAP_SYS_ADMIN to create a new devcgroup, update a devcgroup's access, or move a task to a new devcgroup.

CONFIG_COMMONCAP is defined whenever security/commoncap.c should be compiled, so that the decision of whether to show the option for FILE_CAPABILITIES can be a bit cleaner.

Changelog:

Mar 13 2008: move the dev_cgroup support into capability hooks instead of having it as a separate security module.
Support root_plug with devcg.
Note that due to this change, devcg will not be enforcing if the dummy module is loaded, or if selinux is loaded without capabilities.
Mar 12 2008: allow dev_cgroup lsm to be used when SECURITY=n, and allow stacking with SELinux and Smack. Don't work too hard in Kconfig to prevent a warning when smack+devcg are both compiled in, worry about that later.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

```
include/linux/capability.h    | 11 +-  
include/linux/cgroup_subsys.h |  6 +  
include/linux/devcg.h        | 69 +++++++  
include/linux/security.h     |  7 +-  
init/Kconfig                 |  7 +  
kernel/Makefile              |  1 +  
kernel/dev_cgroup.c          | 411 +++++++++++++++++++++++++++++++  
security/Kconfig              |  6 +-  
security/Makefile             | 12 +-  
security/capability.c        |  2 +  
security/commoncap.c          | 13 ++  
security/dev_cgroup.c         | 83 +++++++  
security/root_plug.c          |  2 +  
security/smack/smack_lsm.c   |  5 +  
14 files changed, 624 insertions(+), 11 deletions(-)  
create mode 100644 include/linux/devcg.h  
create mode 100644 kernel/dev_cgroup.c  
create mode 100644 security/dev_cgroup.c
```

```
diff --git a/include/linux/capability.h b/include/linux/capability.h  
index eaab759..f8ecba1 100644  
--- a/include/linux/capability.h  
+++ b/include/linux/capability.h  
@@ -333,7 +333,16 @@ typedef struct kernel_cap_struct {  
  
#define CAP_MAC_ADMIN      33  
  
-#define CAP_LAST_CAP      CAP_MAC_ADMIN  
/* Allow acting on resources in another namespace. In particular:  
+ * 1. when combined with CAP_MKNOD and dev_cgroup is enabled,
```

```

+ *      allow creation of devices not in the device whitelist.
+ *      2. when combined with CAP_SYS_ADMIN and dev_cgroup is enabled,
+ *      allow editing device cgroup whitelist
+ */
+
+#define CAP_NS_OVERRIDE    34
+
+#define CAP_LAST_CAP      CAP_NS_OVERRIDE

#define cap_valid(x) ((x) >= 0 && (x) <= CAP_LAST_CAP)

diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index 1ddebfc..01e8034 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
#endif

/* */
+
+#ifdef CONFIG_CGROUP_DEV
+SUBSYS(devcg)
#endif
+
+/* */
diff --git a/include/linux/devcg.h b/include/linux/devcg.h
new file mode 100644
index 0000000..32e9f90
--- /dev/null
+++ b/include/linux/devcg.h
@@ -0,0 +1,69 @@
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/list.h>
+#include <linux/security.h>
+
+#include <asm/uaccess.h>
+
#define ACC_MKNOD 1
#define ACC_READ 2
#define ACC_WRITE 4
+
#define DEV_BLOCK 1
#define DEV_CHAR 2
#define DEV_ALL 4 /* this represents all devices */
+
#endif CONFIG_CGROUP_DEV

```

```

+/*
+ * whitelist locking rules:
+ * cgroup_lock() cannot be taken under cgroup->lock.
+ * cgroup->lock can be taken with or without cgroup_lock().
+ *
+ * modifications always require cgroup_lock
+ * modifications to a list which is visible require the
+ * cgroup->lock *and* cgroup_lock()
+ * walking the list requires cgroup->lock or cgroup_lock().
+ *
+ * reasoning: dev_whitelist_copy() needs to kmalloc, so needs
+ * a mutex, which the cgroup_lock() is. Since modifying
+ * a visible list requires both locks, either lock can be
+ * taken for walking the list. Since the wh->spinlock is taken
+ * for modifying a public-accessible list, the spinlock is
+ * sufficient for just walking the list.
+ */
+
+struct dev_whitelist_item {
+ u32 major, minor;
+ short type;
+ short access;
+ struct list_head list;
+};
+
+struct dev_cgroup {
+ struct cgroup_subsys_state css;
+ struct list_head whitelist;
+ spinlock_t lock;
+};
+
+static inline struct dev_cgroup *cgroup_to_devcg(
+ struct cgroup *cgroup)
+{
+ return container_of(cgroup_subsys_state(cgroup, devcg_subsys_id),
+ struct dev_cgroup, css);
+}
+
+extern struct cgroup_subsys devcg_subsys;
+
+extern int devcgroup_inode_permission(struct inode *inode, int mask,
+ struct nameidata *nd);
+extern int devcgroup_inode_mknod(struct inode *dir, struct dentry *dentry,
+ int mode, dev_t dev);
+#else
+static inline int devcgroup_inode_permission(struct inode *inode, int mask,
+ struct nameidata *nd)
+{ return 0; }

```

```

+static inline int devcgroup_inode_mknod(struct inode *dir, struct dentry *dentry,
+    int mode, dev_t dev)
+{ return 0; }
+#endif
diff --git a/include/linux/security.h b/include/linux/security.h
index 2231526..9d562b6 100644
--- a/include/linux/security.h
+++ b/include/linux/security.h
@@ @ -57,6 +57,8 @@ extern int cap_inode_setxattr(struct dentry *dentry, char *name, void
 *value, si
extern int cap_inode_removexattr(struct dentry *dentry, char *name);
extern int cap_inode_need_killpriv(struct dentry *dentry);
extern int cap_inode_killpriv(struct dentry *dentry);
+extern int cap_inode_permission(struct inode *inode, int mask, struct nameidata *nd);
+extern int cap_inode_mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t dev);
extern int cap_task_post_setuid (uid_t old_ruid, uid_t old_euid, uid_t old_suid, int flags);
extern void cap_task_reparent_to_init (struct task_struct *p);
extern int cap_task_kill(struct task_struct *p, struct siginfo *info, int sig, u32 secid);
@@ @ -1735,6 +1737,7 @@ int security_secctx_to_secid(char *secdata, u32 seclen, u32 *secid);
void security_release_secctx(char *secdata, u32 seclen);

#else /* CONFIG_SECURITY */
+
struct security_mnt_opts {
};

@@ @ -2011,7 +2014,7 @@ static inline int security_inode_mknod (struct inode *dir,
    struct dentry *dentry,
    int mode, dev_t dev)
{
- return 0;
+ return cap_inode_mknod(dir, dentry, mode, dev);
}

static inline int security_inode_rename (struct inode *old_dir,
@@ @ -2036,7 +2039,7 @@ static inline int security_inode_follow_link (struct dentry *dentry,
static inline int security_inode_permission (struct inode *inode, int mask,
    struct nameidata *nd)
{
- return 0;
+ return cap_inode_permission(inode, mask, nd);
}

static inline int security_inode_setattr (struct dentry *dentry,
diff --git a/init/Kconfig b/init/Kconfig
index 009f2d8..05343a2 100644
--- a/init/Kconfig
+++ b/init/Kconfig

```

```

@@ -298,6 +298,13 @@ config CGROUP_NS
    for instance virtual servers and checkpoint/restart
    jobs.

+config CGROUP_DEV
+ bool "Device controller for cgroups"
+ depends on CGROUPS && EXPERIMENTAL
+ help
+   Provides a cgroup implementing whitelists for devices which
+   a process in the cgroup can mknod or open.
+
config CPUSETS
bool "Cpuset support"
depends on SMP && CGROUPS
diff --git a/kernel/Makefile b/kernel/Makefile
index 9cc073e..74cd321 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ @ -42,6 +42,7 @@ obj-$(CONFIG_CGROUPS) += cgroup.o
obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
obj-$(CONFIG_CPUSETS) += cpuset.o
obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
+obj-$(CONFIG_CGROUP_DEV) += dev_cgroup.o
obj-$(CONFIG_UTS_NS) += utsname.o
obj-$(CONFIG_USER_NS) += user_namespace.o
obj-$(CONFIG_PID_NS) += pid_namespace.o
diff --git a/kernel/dev_cgroup.c b/kernel/dev_cgroup.c
new file mode 100644
index 0000000..f088824
--- /dev/null
+++ b/kernel/dev_cgroup.c
@@ -0,0 +1,411 @@
+/*
+ * dev_cgroup.c - device cgroup subsystem
+ *
+ * Copyright 2007 IBM Corp
+ */
+
+#include <linux/devcg.h>
+
+static int devcg_can_attach(struct cgroup_subsys *ss,
+    struct cgroup *new_cgroup, struct task_struct *task)
+{
+    struct cgroup *orig;
+
+    if (!capable(CAP_SYS_ADMIN) || !capable(CAP_NS_OVERRIDE))
+        return -EPERM;
+

```

```

+ if (current != task) {
+   if (!cgroup_is_descendant(new_cgroup))
+     return -EPERM;
+ }
+
+ if (atomic_read(&new_cgroup->count) != 0)
+   return -EPERM;
+
+ orig = task_cgroup(task, devcg_subsys_id);
+ if (orig && orig != new_cgroup->parent)
+   return -EPERM;
+
+ return 0;
+}
+
+/*
+ * called under cgroup_lock()
+ */
+int dev_whitelist_copy(struct list_head *dest, struct list_head *orig)
+{
+ struct dev_whitelist_item *wh, *tmp, *new;
+
+ list_for_each_entry(wh, orig, list) {
+   new = kmalloc(sizeof(*wh), GFP_KERNEL);
+   if (!new)
+     goto free_and_exit;
+   new->major = wh->major;
+   new->minor = wh->minor;
+   new->type = wh->type;
+   new->access = wh->access;
+   list_add_tail(&new->list, dest);
+ }
+
+ return 0;
+
+free_and_exit:
+ list_for_each_entry_safe(wh, tmp, dest, list) {
+   list_del(&wh->list);
+   kfree(wh);
+ }
+ return -ENOMEM;
+}
+
+/* Stupid prototype - don't bother combining existing entries */
+/*
+ * called under cgroup_lock()
+ * since the list is visible to other tasks, we need the spinlock also
+ */

```

```

+int dev_whitelist_add(struct dev_cgroup *dev_cgroup,
+    struct dev_whitelist_item *wh)
+{
+    struct dev_whitelist_item *whcopy;
+
+    whcopy = kmalloc(sizeof(*whcopy), GFP_KERNEL);
+    if (!whcopy)
+        return -ENOMEM;
+
+    memcpy(whcopy, wh, sizeof(*whcopy));
+    spin_lock(&dev_cgroup->lock);
+    list_add_tail(&whcopy->list, &dev_cgroup->whitelist);
+    spin_unlock(&dev_cgroup->lock);
+    return 0;
+}
+
+/*
+ * called under cgroup_lock()
+ * since the list is visible to other tasks, we need the spinlock also
+ */
+void dev_whitelist_rm(struct dev_cgroup *dev_cgroup,
+    struct dev_whitelist_item *wh)
+{
+    struct dev_whitelist_item *walk, *tmp;
+
+    spin_lock(&dev_cgroup->lock);
+    list_for_each_entry_safe(walk, tmp, &dev_cgroup->whitelist, list) {
+        if (walk->type == DEV_ALL)
+            goto remove;
+        if (walk->type != wh->type)
+            continue;
+        if (walk->major != ~0 && walk->major != wh->major)
+            continue;
+        if (walk->minor != ~0 && walk->minor != wh->minor)
+            continue;
+
+remove:
+        walk->access &= ~wh->access;
+        if (!walk->access) {
+            list_del(&walk->list);
+            kfree(walk);
+        }
+    }
+    spin_unlock(&dev_cgroup->lock);
+}
+
+/*
+ * Rules: you can only create a cgroup if

```

```

+ * 1. you are capable(CAP_SYS_ADMIN|CAP_NS_OVERRIDE)
+ * 2. the target cgroup is a descendant of your own cgroup
+ *
+ * Note: called from kernel/cgroup.c with cgroup_lock() held.
+ */
+static struct cgroup_subsys_state *devcg_create(struct cgroup_subsys *ss,
+       struct cgroup *cgroup)
+{
+ struct dev_cgroup *dev_cgroup, *parent_dev_cgroup;
+ struct cgroup *parent_cgroup;
+ int ret;
+
+ if (!capable(CAP_SYS_ADMIN) || !capable(CAP_NS_OVERRIDE))
+     return ERR_PTR(-EPERM);
+ if (!cgroup_is_descendant(cgroup))
+     return ERR_PTR(-EPERM);
+
+ dev_cgroup = kzalloc(sizeof(*dev_cgroup), GFP_KERNEL);
+ if (!dev_cgroup)
+     return ERR_PTR(-ENOMEM);
+ INIT_LIST_HEAD(&dev_cgroup->whitelist);
+ parent_cgroup = cgroup->parent;
+
+ if (parent_cgroup == NULL) {
+     struct dev_whitelist_item *wh;
+     wh = kmalloc(sizeof(*wh), GFP_KERNEL);
+     wh->minor = wh->major = ~0;
+     wh->type = DEV_ALL;
+     wh->access = ACC_MKNOD | ACC_READ | ACC_WRITE;
+     list_add(&wh->list, &dev_cgroup->whitelist);
+ } else {
+     parent_dev_cgroup = cgroup_to_devcg(parent_cgroup);
+     ret = dev_whitelist_copy(&dev_cgroup->whitelist,
+           &parent_dev_cgroup->whitelist);
+     if (ret) {
+         kfree(dev_cgroup);
+         return ERR_PTR(ret);
+     }
+ }
+
+ spin_lock_init(&dev_cgroup->lock);
+ return &dev_cgroup->css;
+}

+static void devcg_destroy(struct cgroup_subsys *ss,
+       struct cgroup *cgroup)
+{
+ struct dev_cgroup *dev_cgroup;

```

```

+ struct dev_whitelist_item *wh, *tmp;
+
+ dev_cgroup = cgroup_to_devcg(cgroup);
+ list_for_each_entry_safe(wh, tmp, &dev_cgroup->whitelist, list) {
+ list_del(&wh->list);
+ kfree(wh);
+
+ kfree(dev_cgroup);
+
+
+">#define DEVCG_ALLOW 1
+">#define DEVCG_DENY 2
+
+void set_access(char *acc, short access)
+{
+ int idx = 0;
+ memset(acc, 0, 4);
+ if (access & ACC_READ)
+ acc[idx++] = 'r';
+ if (access & ACC_WRITE)
+ acc[idx++] = 'w';
+ if (access & ACC_MKNOD)
+ acc[idx++] = 'm';
+
+char type_to_char(short type)
+{
+ if (type == DEV_ALL)
+ return 'a';
+ if (type == DEV_CHAR)
+ return 'c';
+ if (type == DEV_BLOCK)
+ return 'b';
+ return 'X';
+
+static void set_majmin(char *str, int len, unsigned m)
+{
+ memset(str, 0, len);
+ if (m == ~0)
+ sprintf(str, "*");
+ else
+ sprintf(str, len, "%d", m);
+
+char *print_whitelist(struct dev_cgroup *devcggroup, int *len)
+{
+ char *buf, *s, acc[4];

```

```

+ struct dev_whitelist_item *wh;
+ int ret;
+ int count = 0;
+ char maj[10], min[10];
+
+ buf = kmalloc(4096, GFP_KERNEL);
+ if (!buf)
+   return ERR_PTR(-ENOMEM);
+ s = buf;
+ *s = '\0';
+ *len = 0;
+
+ spin_lock(&devcgroup->lock);
+ list_for_each_entry(wh, &devcgroup->whitelist, list) {
+   set_access(acc, wh->access);
+   set_majmin(maj, 10, wh->major);
+   set_majmin(min, 10, wh->minor);
+   ret = snprintf(s, 4095-(s-buf), "%c %s %s %s\n",
+     type_to_char(wh->type), maj, min, acc);
+   if (s+ret >= buf+4095) {
+     kfree(buf);
+     buf = ERR_PTR(-ENOMEM);
+     break;
+   }
+   s += ret;
+   *len += ret;
+   count++;
+ }
+ spin_unlock(&devcgroup->lock);
+
+ return buf;
+}
+
+static ssize_t devcg_access_read(struct cgroup *cgroup,
+  struct cftype *cft, struct file *file,
+  char __user *userbuf, size_t nbytes, loff_t *ppos)
+{
+ struct dev_cgroup *devcgrp = cgroup_to_devcg(cgroup);
+ int filetype = cft->private;
+ char *buffer;
+ int len, retval;
+
+ if (filetype != DEVCG_ALLOW)
+   return -EINVAL;
+ buffer = print_whitelist(devcgrp, &len);
+ if (IS_ERR(buffer))
+   return PTR_ERR(buffer);
+

```

```

+ retval = simple_read_from_buffer(userbuf, nbytes, ppos, buffer, len);
+ kfree(buffer);
+ return retval;
+}
+
+static inline short convert_access(char *acc)
+{
+ short access = 0;
+
+ while (*acc) {
+ switch (*acc) {
+ case 'r':
+ case 'R': access |= ACC_READ; break;
+ case 'w':
+ case 'W': access |= ACC_WRITE; break;
+ case 'm':
+ case 'M': access |= ACC_MKNOD; break;
+ case '\n': break;
+ default:
+ return -EINVAL;
+ }
+ acc++;
+ }
+
+ return access;
+}
+
+static inline short convert_type(char intype)
+{
+ short type = 0;
+ switch (intype) {
+ case 'a': type = DEV_ALL; break;
+ case 'c': type = DEV_CHAR; break;
+ case 'b': type = DEV_BLOCK; break;
+ default: type = -EACCES; break;
+ }
+ return type;
+}
+
+static int convert_majmin(char *m, unsigned *u)
+{
+ if (m[0] == '*') {
+ *u = ~0;
+ return 0;
+ }
+ if (sscanf(m, "%u", u) != 1)
+ return -EINVAL;
+ return 0;

```

```

+}
+
+static ssize_t devcg_access_write(struct cgroup *cgroup, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+    struct cgroup *cur_cgroup;
+    struct dev_cgroup *devcgrp, *cur_devcgroup;
+    int filetype = cft->private;
+    char *buffer, acc[4], maj[10], min[10];
+    int retval = 0;
+    int nitems;
+    char type;
+    struct dev_whitelist_item wh;
+
+    if (!capable(CAP_SYS_ADMIN) || !capable(CAP_NS_OVERRIDE))
+        return -EPERM;
+
+    devcgrp = cgroup_to_devcg(cgroup);
+    cur_cgroup = task_cgroup(current, devcg_subsys.subsys_id);
+    cur_devcgroup = cgroup_to_devcg(cur_cgroup);
+
+    buffer = kmalloc(nbytes+1, GFP_KERNEL);
+    if (!buffer)
+        return -ENOMEM;
+
+    if (copy_from_user(buffer, userbuf, nbytes)) {
+        retval = -EFAULT;
+        goto out1;
+    }
+    buffer[nbytes] = 0; /* nul-terminate */
+
+    cgroup_lock();
+    if (cgroup_is_removed(cgroup)) {
+        retval = -ENODEV;
+        goto out2;
+    }
+
+    memset(&wh, 0, sizeof(wh));
+    memset(acc, 0, 4);
+    nitems = sscanf(buffer, "%c %9s %9s %3s", &type, maj, min,
+        acc);
+    if (nitems != 4)
+        goto out2;
+    wh.type = convert_type(type);
+    if (wh.type < 0)
+        goto out2;

```

```

+ wh.access = convert_access(acc);
+ if (convert_majmin(maj, &wh.major))
+ goto out2;
+ if (convert_majmin(min, &wh.minor))
+ goto out2;
+ if (wh.access < 0)
+ goto out2;
+ retval = 0;
+ switch (filetype) {
+ case DEVCG_ALLOW:
+ retval = dev_whitelist_add(devcgrp, &wh);
+ break;
+ case DEVCG_DENY:
+ dev_whitelist_rm(devcgrp, &wh);
+ break;
+ default:
+ retval = -EINVAL;
+ goto out2;
+ }
+
+ if (retval == 0)
+ retval = nbytes;
+
+out2:
+ cgroup_unlock();
+out1:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype dev_cgroup_files[] = {
+ {
+ .name = "allow",
+ .read = devcg_access_read,
+ .write = devcg_access_write,
+ .private = DEVCG_ALLOW,
+ },
+ {
+ .name = "deny",
+ .write = devcg_access_write,
+ .private = DEVCG_DENY,
+ },
+};
+
+static int devcg_populate(struct cgroup_subsys *ss,
+ struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss, dev_cgroup_files,

```

```

+     ARRAY_SIZE(dev_cgroup_files));
+
+
+struct cgroup_subsys devcg_subsys = {
+ .name = "devcg",
+ .can_attach = devcg_can_attach,
+ .create = devcg_create,
+ .destroy = devcg_destroy,
+ .populate = devcg_populate,
+ .subsys_id = devcg_subsys_id,
+};
diff --git a/security/Kconfig b/security/Kconfig
index 5dfc206..c7960c0 100644
--- a/security/Kconfig
+++ b/security/Kconfig
@@ -81,9 +81,13 @@ config SECURITY_CAPABILITIES
    This enables the "default" Linux capabilities functionality.
    If you are unsure how to answer this question, answer Y.

+config COMMONCAP
+ bool
+ default !SECURITY || SECURITY_CAPABILITIES || SECURITY_ROOTPLUG ||
SECURITY_SMACK || CGROUP_DEV
+
config SECURITY_FILE_CAPABILITIES
    bool "File POSIX Capabilities (EXPERIMENTAL)"
- depends on (SECURITY=n || SECURITY_CAPABILITIES!=n) && EXPERIMENTAL
+ depends on COMMONCAP && EXPERIMENTAL
    default n
    help
        This enables filesystem capabilities, allowing you to give
diff --git a/security/Makefile b/security/Makefile
index 9e8b025..6093003 100644
--- a/security/Makefile
+++ b/security/Makefile
@@ -6,15 +6,13 @@ obj-$(CONFIG_KEYS) += keys/
subdir-$(CONFIG_SECURITY_SELINUX) += selinux
subdir-$(CONFIG_SECURITY_SMACK) += smack

-# if we don't select a security model, use the default capabilities
-ifneq ($(CONFIG_SECURITY),y)
-obj-y += commoncap.o
-endif
+obj-$(CONFIG_COMMONCAP) += commoncap.o

# Object file lists
obj-$(CONFIG_SECURITY) += security.o dummy.o inode.o
# Must precede capability.o in order to stack properly.

```

```

obj-$(CONFIG_SECURITY_SELINUX) += selinux/built-in.o
-obj-$(CONFIG_SECURITY_SMACK) += commoncap.o smack/built-in.o
-obj-$(CONFIG_SECURITY_CAPABILITIES) += commoncap.o capability.o
-obj-$(CONFIG_SECURITY_ROOTPLUG) += commoncap.o root_plug.o
+obj-$(CONFIG_SECURITY_SMACK) += smack/built-in.o
+obj-$(CONFIG_SECURITY_CAPABILITIES) += capability.o
+obj-$(CONFIG_SECURITY_ROOTPLUG) += root_plug.o
+obj-$(CONFIG_CGROUP_DEV) += dev_cgroup.o
diff --git a/security/capability.c b/security/capability.c
index 8340655..1202991 100644
--- a/security/capability.c
+++ b/security/capability.c
@@ @ -39,6 +39,8 @@ static struct security_operations capability_ops = {
    .inode_removexattr = cap_inode_removexattr,
    .inode_need_killpriv = cap_inode_need_killpriv,
    .inode_killpriv = cap_inode_killpriv,
+   .inode_permission = cap_inode_permission,
+   .inode_mknod = cap_inode_mknod,
    .task_kill = cap_task_kill,
    .task_setscheduler = cap_task_setscheduler,
diff --git a/security/commoncap.c b/security/commoncap.c
index 83f2691..68c6a97 100644
--- a/security/commoncap.c
+++ b/security/commoncap.c
@@ @ -26,6 +26,7 @@
 #include <linux/sched.h>
 #include <linux/prctl.h>
 #include <linux/securebits.h>
+#include <linux/devcg.h>

int cap_netlink_send(struct sock *sk, struct sk_buff *skb)
{
@@ @ -160,6 +161,18 @@ static inline void bprm_clear_caps(struct linux_binprm *bprm)
    bprm->cap_effective = false;
}

+int cap_inode_permission(struct inode *inode, int mask,
+    struct nameidata *nd)
+{
+    return devcgroup_inode_permission(inode, mask, nd);
+}
+
+int cap_inode_mknod(struct inode *dir, struct dentry *dentry,
+    int mode, dev_t dev)
+{
+    return devcgroup_inode_mknod(dir, dentry, mode, dev);
+}

```

```

+
#define CONFIG_SECURITY_FILE_CAPABILITIES

int cap_inode_need_killpriv(struct dentry *dentry)
diff --git a/security/dev_cgroup.c b/security/dev_cgroup.c
new file mode 100644
index 0000000..eb65411
--- /dev/null
+++ b/security/dev_cgroup.c
@@ -0,0 +1,83 @@
+/*
+ * LSM portion of the device cgroup subsystem.
+ */
+ * Copyright 2007 IBM Corp
+ */
+
+#include <linux/devcg.h>
+
+int devcgroup_inode_permission(struct inode *inode, int mask,
+       struct nameidata *nd)
+{
+    struct cgroup *cgroup;
+    struct dev_cgroup *dev_cgroup;
+    struct dev_whitelist_item *wh;
+
+    dev_t device = inode->i_rdev;
+    if (!device)
+        return 0;
+    if (!S_ISBLK(inode->i_mode) && !S_ISCHR(inode->i_mode))
+        return 0;
+    cgroup = task_cgroup(current, devcg_subsys.subsys_id);
+    dev_cgroup = cgroup_to_devcg(cgroup);
+    if (!dev_cgroup)
+        return 0;
+
+    spin_lock(&dev_cgroup->lock);
+    list_for_each_entry(wh, &dev_cgroup->whitelist, list) {
+        if (wh->type & DEV_ALL)
+            goto acc_check;
+        if ((wh->type & DEV_BLOCK) && !S_ISBLK(inode->i_mode))
+            continue;
+        if ((wh->type & DEV_CHAR) && !S_ISCHR(inode->i_mode))
+            continue;
+        if (wh->major != ~0 && wh->major != imajor(inode))
+            continue;
+        if (wh->minor != ~0 && wh->minor != iminor(inode))
+            continue;
+    acc_check:

```

```

+ if ((mask & MAY_WRITE) && !(wh->access & ACC_WRITE))
+ continue;
+ if ((mask & MAY_READ) && !(wh->access & ACC_READ))
+ continue;
+ spin_unlock(&dev_cgroup->lock);
+ return 0;
+
+ }
+ spin_unlock(&dev_cgroup->lock);
+
+
+ return -EPERM;
+}
+
+int devcgroup_inode_mknod(struct inode *dir, struct dentry *dentry,
+    int mode, dev_t dev)
+{
+ struct cgroup *cgroup;
+ struct dev_cgroup *dev_cgroup;
+ struct dev_whitelist_item *wh;
+
+ cgroup = task_cgroup(current, devcg_subsys.subsys_id);
+ dev_cgroup = cgroup_to_devcg(cgroup);
+ if (!dev_cgroup)
+ return 0;
+
+ spin_lock(&dev_cgroup->lock);
+ list_for_each_entry(wh, &dev_cgroup->whitelist, list) {
+ if (wh->type & DEV_ALL)
+ goto acc_check;
+ if ((wh->type & DEV_BLOCK) && !S_ISBLK(mode))
+ continue;
+ if ((wh->type & DEV_CHAR) && !S_ISCHR(mode))
+ continue;
+ if (wh->major != ~0 && wh->major != MAJOR(dev))
+ continue;
+ if (wh->minor != ~0 && wh->minor != MINOR(dev))
+ continue;
+acc_check:
+ if (!(wh->access & ACC_MKNOD))
+ continue;
+ spin_unlock(&dev_cgroup->lock);
+ return 0;
+
+ }
+ spin_unlock(&dev_cgroup->lock);
+ return -EPERM;
+}
diff --git a/security/root_plug.c b/security/root_plug.c
index a41cf42..090015d 100644
--- a/security/root_plug.c

```

```

+++ b/security/root_plug.c
@@ -80,6 +80,8 @@ static struct security_operations rootplug_security_ops = {
 .capset_check = cap_capset_check,
 .capset_set = cap_capset_set,
 .capable = cap_capable,
+ .inode_mknod = cap_inode_mknod,
+ .inode_permission = cap_inode_permission,
 
 .bprm_apply_creds = cap_bprm_apply_creds,
 .bprm_set_security = cap_bprm_set_security,
diff --git a/security/smack/smack_lsm.c b/security/smack/smack_lsm.c
index 20ec35c..6814aaa 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -523,6 +523,10 @@ static int smack_inode_rename(struct inode *old_inode,
 static int smack_inode_permission(struct inode *inode, int mask,
         struct nameidata *nd)
{
+ int err;
+ err = cap_inode_permission(inode, mask, nd);
+ if (err)
+     return err;
 /*
 * No permission to check. Existence test. Yup, it's there.
 */
@@ -2460,6 +2464,7 @@ struct security_operations smack_ops = {
 .inode_getsecurity = smack_inode_getsecurity,
 .inode_setsecurity = smack_inode_setsecurity,
 .inode_listsecurity = smack_inode_listsecurity,
+ .inode_mknod = cap_inode_mknod,
 
 .file_permission = smack_file_permission,
 .file_alloc_security = smack_file_alloc_security,
--
```

1.5.1

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