Subject: [PATCH 1/2][RFC] containers: improve automatic container naming
Posted by serue on Fri, 01 Jun 2007 21:48:09 GMT
View Forum Message <> Reply to Message

This compiles and boots, but is not intended for inclusion in -mm (yet),
just as an RFC for the naming scheme to fix the bug Andrew pointed out.

Seem ok overall?

thanks,
-serge

>From 8e9b972f7482415777e982d3bc9a0d55cbaf862b Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Fri, 1 Jun 2007 15:32:15 -0400
Subject: [PATCH 1/2] containers: improve automatic container naming

The automatic naming of containers created using container_clone()
is currently broken (not protected from wraparound) and inconvenient.

Add a per-container counter for use in naming children of the container.
Before two unshares in a row by one process, and a third in another,
would result in

 /node1/node2
 /node3

The current scheme should result in

 /node1/node1
 /node2

Also, keep a hash table populated with used names, to protect
against counter wrap-around.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/container.h | 8 +++
 kernel/container.c | 116 ++--
 2 files changed, 120 insertions(+), 4 deletions(-)

diff --git a/include/linux/container.h b/include/linux/container.h
index 37c0bdf..61c750d 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -75,6 +75,14 @@ struct container {
  atomic_t count;

```
  /*
+  * these are to support automatic naming of automatically
+  * created containers
+  */
+ int auto_cnt_set;  /* 1 if thie container is using up a auto_cnt */
+ unsigned long auto_cnt; /* the auto_cnt it's using up */
+ unsigned long next_childcnt; /* next avail cnt for child containers */
+
+ /*
   * We link our 'sibling' struct into our parent's 'children'.
   * Our children link their 'sibling' into our 'children'.
   */
diff --git a/kernel/container.c b/kernel/container.c
index 6f80487..2b2f6f2 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -55,6 +55,7 @@
 #include <linux/time.h>
 #include <linux/backing-dev.h>
 #include <linux/sort.h>
+#include <linux/hash.h>

 #include <asm/uaccess.h>
 #include <asm/atomic.h>
@@ -1819,6 +1820,9 @@ static long container_create(struct container *parent, struct dentry
*dentry,
  INIT_LIST_HEAD(&cont->css_groups);
  INIT_LIST_HEAD(&cont->release_list);

+ cont->auto_cnt_set = 0;
+ cont->next_childcnt = 0;
+
  cont->parent = parent;
  cont->root = parent->root;
  cont->top_container = parent->top_container;
@@ -1894,6 +1898,71 @@ static inline int container_has_css_refs(struct container *cont)
  return 0;
 }

+#define cnthash_shift 3
+#define childcnt_hash(ptr, nr) hash_long(nr+(unsigned long)ptr, cnthash_shift)
+static DEFINE_SPINLOCK(childcntlock);
+
+struct container_child_hash_elem {
+ struct container *parent;
+ unsigned long cnt;
+ struct hlist_node list;
+};
```

```
+
+static struct hlist_head cntnum_hash_list[1 << cnthash_shift];
+
+static void init_cntnum_hash(void)
+{
+ int i;
+
+ for (i=0; i<1 << cnthash_shift; i++)
+  INIT_HLIST_HEAD(&cntnum_hash_list[i]);
+}
+
+static inline struct container_child_hash_elem *find_cntnum_hash(
+  struct container *parent, unsigned long count)
+{
+ struct container_child_hash_elem *he;
+ struct hlist_head *h;
+ struct hlist_node *tmp;
+
+ h = &cntnum_hash_list[childcnt_hash(parent, count)];
+ hlist_for_each_entry(he, tmp, h, list) {
+  if (he->parent == parent && he->cnt == count)
+   return he;
+ }
+ return NULL;
+}
+
+#define UNSET_HASH 0
+#define SET_HASH 1
+static inline int set_cntnum_hash(struct container *parent,
+        unsigned long count, int set)
+{
+ struct container_child_hash_elem *he;
+ int idx;
+
+ he = find_cntnum_hash(parent, count);
+ if (he) {
+  if (!set) {
+   hlist_del(&he->list);
+   kfree(he);
+   return 0;
+  }
+  return -ENOENT;
+ }
+ if (!set)
+  return -ENOENT;
+ he = kmalloc(sizeof(*he), GFP_KERNEL);
+ if (!he)
+  return -ENOMEM;
```

```
+ he->parent = parent;
+ he->cnt = count;
+ INIT_HLIST_NODE(&he->list);
+ idx = childcnt_hash(parent, count);
+ hlist_add_head(&he->list, &cntnum_hash_list[idx]);
+ return 0;
+}
+
 static int container_rmdir(struct inode *unused_dir, struct dentry *dentry)
 {
  struct container *cont = dentry->d_fsdata;
@@ -1941,6 +2010,15 @@ static int container_rmdir(struct inode *unused_dir, struct dentry
*dentry)
  dput(d);
  root->number_of_containers--;

+ if (cont->auto_cnt_set) {
+  spin_lock(&childcntlock);
+  if (set_cntnum_hash(parent, cont->auto_cnt, UNSET_HASH))
+   printk(KERN_NOTICE "%s: could not unset %lu\n",
+    __FUNCTION__, cont->auto_cnt);
+  cont->auto_cnt_set = 0;
+  spin_unlock(&childcntlock);
+ }
+
  if (!list_empty(&cont->release_list))
   list_del(&cont->release_list);
  set_bit(CONT_RELEASABLE, &parent->flags);
@@ -2063,6 +2141,8 @@ int __init container_init(void)
  err = register_filesystem(&container_fs_type);
  if (err < 0)
   goto out;
+
+ init_cntnum_hash();

  entry = create_proc_entry("containers", 0, NULL);
  if (entry)
@@ -2302,10 +2382,31 @@ void container_exit(struct task_struct *tsk, int run_callbacks)
   put_css_group_taskexit(cg);
 }

-static atomic_t namecnt;
-static void get_unused_name(char *buf)
+/*
+ * the while (find_cntnum_hash) loop will only be hit after
+ * we wrap around on next_childnct, which should never happen.
+ */
+static int get_unused_name(struct container *parent, unsigned long *out_cnt,
```

```
+  char *buf)
 {
- sprintf(buf, "node%d", atomic_inc_return(&namecnt));
+ unsigned long cnt;
+ int err = 0;
+
+ spin_lock(&childcntlock);
+ cnt = parent->next_childcnt;
+ while (find_cntnum_hash(parent, cnt))
+  cnt++;
+ err = set_cntnum_hash(parent, cnt, SET_HASH);
+ if (err)
+  goto out_unlock;
+
+ sprintf(buf, "node%lu", cnt);
+ parent->next_childcnt = cnt+1;
+ *out_cnt = cnt;
+
+out_unlock:
+ spin_unlock(&childcntlock);
+ return err;
 }

 /**
@@ -2318,6 +2419,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  struct dentry *dentry;
  int ret = 0;
  char nodename[32];
+ unsigned long auto_cntnum;
  struct container *parent, *child;
  struct inode *inode;
  struct css_group *cg;
@@ -2348,7 +2450,10 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  mutex_unlock(&container_mutex);

  /* Now do the VFS work to create a container */
- get_unused_name(nodename);
+ ret = get_unused_name(parent, &auto_cntnum, nodename);
+ if (ret)
+  goto out_container_mutex;
+
  inode = parent->dentry->d_inode;

  /* Hold the parent directory mutex across this operation to
@@ -2380,6 +2485,8 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
```

```
   ret = -ENOMEM;
   goto out_release;
  }
+ child->auto_cnt_set = 1;
+ child->auto_cnt = auto_cntnum;

  /* The container now exists. Retake container_mutex and check
   * that we're still in the same state that we thought we
@@ -2408,6 +2515,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  out_release:
  mutex_unlock(&inode->i_mutex);

+ out_container_mutex:
  mutex_lock(&container_mutex);
  put_css_group(cg);
  mutex_unlock(&container_mutex);
--
1.5.1.1.GIT
```

Subject: [PATCH 2/2][RFC] containers interface to nsproxy subsystem
Posted by serue on Fri, 01 Jun 2007 21:49:43 GMT
View Forum Message <> Reply to Message

Again, not compiles and boots, but not sufficiently tested for
inclusion.

thanks,
-serge

```
>From 6c03518dbda04bf20a9ddeefb291bf1c63e32ec1 Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Fri, 1 Jun 2007 17:30:16 -0400
Subject: [PATCH 2/2] Container interface to nsproxy subsystem

When a task enters a new namespace via a clone() or unshare(), a new
container is created and the task moves into it.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/container_subsys.h |    6 ++
 include/linux/nsproxy.h          |    7 +++
 init/Kconfig                     |    9 ++++
 kernel/Makefile                  |    1 +
 kernel/container.c               |   15 +++++-
 kernel/ns_container.c            |   96 ++++++++++++++++++++++++++++++++++++++
 kernel/nsproxy.c                 |   16 ++++++
```

```
 7 files changed, 148 insertions(+), 2 deletions(-)
 create mode 100644 kernel/ns_container.c

diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
index 8fea7cf..9861751 100644
--- a/include/linux/container_subsys.h
+++ b/include/linux/container_subsys.h
@@ -24,3 +24,9 @@ SUBSYS(debug)
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_NS
+SUBSYS(ns)
+#endif
+
+/* */
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 189e0dc..8be975b 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -54,4 +54,11 @@ static inline void exit_task_namespaces(struct task_struct *p)
   put_nsproxy(ns);
  }
 }
+
+#ifdef CONFIG_CONTAINER_NS
+int ns_container_clone(struct task_struct *tsk);
+#else
+static inline int ns_container_clone(struct task_struct *tsk) { return 0; }
+#endif
+
 #endif
diff --git a/init/Kconfig b/init/Kconfig
index 5861ad9..d79c505 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -355,6 +355,15 @@ config CONTAINER_CPUACCT
    Provides a simple Resource Controller for monitoring the
    total CPU consumed by the tasks in a container

+config CONTAINER_NS
+       bool "Namespace container subsystem"
+       select CONTAINERS
+       help
+         Provides a simple namespace container subsystem to
+         provide hierarchical naming of sets of namespaces,
+         for instance virtual servers and checkpoint/restart
```

```
+          jobs.
+
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
diff --git a/kernel/Makefile b/kernel/Makefile
index f73b3d3..34f2345 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -40,6 +40,7 @@ obj-$(CONFIG_CONTAINERS) += container.o
 obj-$(CONFIG_CONTAINER_DEBUG) += container_debug.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
+obj-$(CONFIG_CONTAINER_NS) += ns_container.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
diff --git a/kernel/container.c b/kernel/container.c
index 2b2f6f2..ac5879f 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -2523,14 +2523,25 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  return ret;
 }

-/* See if "cont" is a descendant of the current task's container in
- * the appropriate hierarchy */
+/*
+ * See if "cont" is a descendant of the current task's container in
+ * the appropriate hierarchy
+ *
+ * If we are sending in dummytop, then presumably we are creating
+ * the top container in the subsystem.
+ *
+ * Called only by the ns (nsproxy) container.
+ */

 int container_is_descendant(const struct container *cont)
 {
  int ret;
  struct container *target;
  int subsys_id;
+
+ if (cont == dummytop)
+  return 1;
+
  get_first_subsys(cont, NULL, &subsys_id);
```

```
  target = task_container(current, subsys_id);
  while (cont != target && cont!= cont->top_container) {
diff --git a/kernel/ns_container.c b/kernel/ns_container.c
new file mode 100644
index 0000000..afa50d9
--- /dev/null
+++ b/kernel/ns_container.c
@@ -0,0 +1,96 @@
+/*
+ * ns_container.c - namespace container subsystem
+ *
+ * Copyright 2006, 2007 IBM Corp
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+
+struct nscont {
+ struct container_subsys_state css;
+ spinlock_t lock;
+};
+
+struct container_subsys ns_subsys;
+
+static inline struct nscont *container_nscont(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, ns_subsys_id),
+       struct nscont, css);
+}
+
+int ns_container_clone(struct task_struct *tsk)
+{
+ return container_clone(tsk, &ns_subsys);
+}
+
+/*
+ * Rules:
+ *   1. you can only enter a container which is a child of your current
+ *      container
+ *   2. you can only place another process into a container if
+ *      a. you have CAP_SYS_ADMIN
+ *      b. your container is an ancestor of tsk's destination container
+ *        (hence either you are in the same container as tsk, or in an
+ *         ancestor container thereof)
+ */
+int ns_can_attach(struct container_subsys *ss,
+    struct container *cont, struct task_struct *tsk)
```

```
+{
+ struct container *c;
+
+ if (current != tsk) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  if (!container_is_descendant(cont))
+   return -EPERM;
+ }
+
+ if (atomic_read(&cont->count) != 0)
+  return -EPERM;
+
+ c = task_container(tsk, ns_subsys_id);
+ if (c && c != cont->parent)
+  return -EPERM;
+
+ return 0;
+}
+
+/*
+ * Rules: you can only create a container if
+ *     1. you are capable(CAP_SYS_ADMIN)
+ *     2. the target container is a descendant of your own container
+ */
+static int ns_create(struct container_subsys *ss, struct container *cont)
+{
+ struct nscont *ns;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return -EPERM;
+ if (!container_is_descendant(cont))
+  return -EPERM;
+
+ ns = kzalloc(sizeof(*ns), GFP_KERNEL);
+ if (!ns) return -ENOMEM;
+ spin_lock_init(&ns->lock);
+ cont->subsys[ns_subsys.subsys_id] = &ns->css;
+ return 0;
+}
+
+static void ns_destroy(struct container_subsys *ss,
+         struct container *cont)
+{
+ struct nscont *ns = container_nscont(cont);
+ kfree(ns);
+}
```

```
+
+struct container_subsys ns_subsys = {
+ .name = "ns",
+ .create = ns_create,
+ .destroy  = ns_destroy,
+ .can_attach = ns_can_attach,
+ .subsys_id = ns_subsys_id,
+};
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 1bc4b55..afce808 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -124,7 +124,14 @@ int copy_namespaces(int flags, struct task_struct *tsk)
   goto out;
  }

+ err = ns_container_clone(tsk);
+ if (err) {
+  put_nsproxy(new_ns);
+  goto out;
+ }
+
  tsk->nsproxy = new_ns;
+
 out:
  put_nsproxy(old_ns);
  return err;
@@ -177,6 +184,15 @@ int unshare_nsproxy_namespaces(unsigned long unshare_flags,
  if (IS_ERR(*new_nsp)) {
   err = PTR_ERR(*new_nsp);
   put_nsproxy(old_ns);
+  goto out;
+ }
+
+ err = ns_container_clone(current);
+ if (err) {
+  put_nsproxy(*new_nsp);
+  put_nsproxy(old_ns);
  }
+
+out:
  return err;
 }
--
1.5.1.1.GIT
```

Subject: Re: [PATCH 1/2][RFC] containers: improve automatic container naming
Posted by Andrew Morton on Fri, 01 Jun 2007 22:23:34 GMT
View Forum Message <> Reply to Message

On Fri, 1 Jun 2007 16:48:09 -0500
"Serge E. Hallyn" <serue@us.ibm.com> wrote:

> This compiles and boots, but is not intended for inclusion in -mm (yet),
> just as an RFC for the naming scheme to fix the bug Andrew pointed out.
>
> Seem ok overall?
>
> thanks,
> -serge
>
> >From 8e9b972f7482415777e982d3bc9a0d55cbaf862b Mon Sep 17 00:00:00 2001
> From: Serge E. Hallyn <serue@us.ibm.com>
> Date: Fri, 1 Jun 2007 15:32:15 -0400
> Subject: [PATCH 1/2] containers: improve automatic container naming
>
> The automatic naming of containers created using container_clone()
> is currently broken (not protected from wraparound) and inconvenient.
>
> Add a per-container counter for use in naming children of the container.
> Before two unshares in a row by one process, and a third in another,
> would result in
>
> /node1/node2
> /node3
>
> The current scheme should result in
>
> /node1/node1
> /node2
>
> Also, keep a hash table populated with used names, to protect
> against counter wrap-around.
>
> ...
>
> include/linux/container.h | 8 +++
> kernel/container.c | 116 ++--

gad, what's all this stuff?

I think an IDR tree would get you what you're after in much less code.
Although it means that container IDs would get recycled quickly across a
remove+add.

Be aware that there are IDR enhancements in Greg's driver tree (and hence
in -mm) which are relevant to this application.

> + if (cont->auto_cnt_set) {

Can we please stop using "cnt" and "cont" to refer to containers? Let's
use "container", OK?

Subject: Re: [PATCH 1/2][RFC] containers: improve automatic container naming
Posted by serue on Fri, 01 Jun 2007 22:38:55 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Fri, 1 Jun 2007 16:48:09 -0500
> "Serge E. Hallyn" <serue@us.ibm.com> wrote:
>
> > This compiles and boots, but is not intended for inclusion in -mm (yet),
> > just as an RFC for the naming scheme to fix the bug Andrew pointed out.
> >
> > Seem ok overall?
> >
> > thanks,
> > -serge
> >
> > >From 8e9b972f7482415777e982d3bc9a0d55cbaf862b Mon Sep 17 00:00:00 2001
> > From: Serge E. Hallyn <serue@us.ibm.com>
> > Date: Fri, 1 Jun 2007 15:32:15 -0400
> > Subject: [PATCH 1/2] containers: improve automatic container naming
> >
> > The automatic naming of containers created using container_clone()
> > is currently broken (not protected from wraparound) and inconvenient.
> >
> > Add a per-container counter for use in naming children of the container.
> > Before two unshares in a row by one process, and a third in another,
> > would result in
> >
> > /node1/node2
> > /node3
> >
> > The current scheme should result in
> >
> > /node1/node1
> > /node2
> >
> > Also, keep a hash table populated with used names, to protect
> > against counter wrap-around.

> >
> > ...
> >
> > include/linux/container.h | 8 +++
> > kernel/container.c | 116 ++--
>
> gad, what's all this stuff?

The sound of me starting over.

> I think an IDR tree would get you what you're after in much less code.

Ok, will look at that. Thanks for the tip!

> Although it means that container IDs would get recycled quickly across a
> remove+add.
>
> Be aware that there are IDR enhancements in Greg's driver tree (and hence
> in -mm) which are relevant to this application.
>
>
> > + if (cont->auto_cnt_set) {
>
> Can we please stop using "cnt" and "cont" to refer to containers? Let's
> use "container", OK?

Ok.

thanks,
-serge