Subject: [PATCH] new cgroup controller "fork"
Posted by Max Kellermann on Thu, 17 Feb 2011 13:31:52 GMT
View Forum Message <> Reply to Message

Can limit the number of fork()/clone() calls in a cgroup. It is
useful as a safeguard against fork bombs.

Signed-off-by: Max Kellermann <mk@cm4all.com>

Documentation/cgroups/fork.txt | 30 +++++++
include/linux/cgroup_fork.h | 26 ++++++
include/linux/cgroup_subsys.h | 6 +
init/Kconfig | 6 +
kernel/Makefile | 1
kernel/cgroup_fork.c | 180 ++
kernel/fork.c | 5 +
7 files changed, 254 insertions(+), 0 deletions(-)
create mode 100644 Documentation/cgroups/fork.txt
create mode 100644 include/linux/cgroup_fork.h
create mode 100644 kernel/cgroup_fork.c

diff --git a/Documentation/cgroups/fork.txt b/Documentation/cgroups/fork.txt
new file mode 100644
index 0000000..dfbf291
--- /dev/null
+++ b/Documentation/cgroups/fork.txt
@@ -0,0 +1,30 @@
+The "fork" Controller
+----------
+
+The "fork" controller limits the number of times a new child process
+or thread can be created. It maintains a per-group counter which gets
+decremented on each fork() / clone(). When the counter reaches zero,
+no process in the cgroup is allowed to create new child
+processes/threads, even if existing ones quit.
+
+This has been proven useful in a shared hosting environment. A new
+temporary cgroup is created for each CGI process, and the maximum fork
+count is configured to a sensible value. Since CGIs are expected to
+run for only a short time with predictable resource usage, this may be
+an appropriate tool to limit the damage that a freaked CGI can do.
+
+Initially, the counter is set to -1, which is a magic value for
+"disabled" - no limits are imposed on the processes in the group. To
+set a new value, type (in the working directory of that control
+group):
+
+ echo 16 > fork.remaining

```
+
+This examples allows 16 forks in the control group.  0 means no
+further forks are allowed.  The limit may be lowered or increased or
+even disabled at any time by a process with write permissions to the
+attribute.
+
+To check if a fork is allowed, the controller walks the cgroup
+hierarchy up, and verifies all ancestors.  The counter of all
+ancestors is decreased.
diff --git a/include/linux/cgroup_fork.h b/include/linux/cgroup_fork.h
new file mode 100644
index 0000000..4ac66b3
--- /dev/null
+++ b/include/linux/cgroup_fork.h
@@ -0,0 +1,26 @@
+#ifndef _LINUX_CGROUP_FORK_H
+#define _LINUX_CGROUP_FORK_H
+
+#ifdef CONFIG_CGROUP_FORK
+
+/**
+ * Checks if another fork is allowed.  Call this before creating a new
+ * child process.
+ *
+ * @return 0 on success, a negative errno value if forking should be
+ * denied
+ */
+int
+cgroup_fork_pre_fork(void);
+
+#else /* !CONFIG_CGROUP_FORK */
+
+static inline int
+cgroup_fork_pre_fork(void)
+{
+ return 0;
+}
+
+#endif /* !CONFIG_CGROUP_FORK */
+
+#endif /* !_LINUX_CGROUP_FORK_H */
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index ccefff0..8ead7f9 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -66,3 +66,9 @@ SUBSYS(blkio)
 #endif
```

```
 /* */
+
+#ifdef CONFIG_CGROUP_FORK
+SUBSYS(fork)
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index 17e2cfb..ef53a85 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -596,6 +596,12 @@ config CGROUP_FREEZER
    Provides a way to freeze and unfreeze all tasks in a
    cgroup.

+config CGROUP_FORK
+ bool "fork controller for cgroups"
+ help
+   Limits the number of fork() calls in a cgroup.  An application
+   for this is to make a cgroup safe against fork bombs.
+
 config CGROUP_DEVICE
  bool "Device controller for cgroups"
  help
diff --git a/kernel/Makefile b/kernel/Makefile
index 353d3fe..b58cc01 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -61,6 +61,7 @@ obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
+obj-$(CONFIG_CGROUP_FORK) += cgroup_fork.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
 obj-$(CONFIG_UTS_NS) += utsname.o
diff --git a/kernel/cgroup_fork.c b/kernel/cgroup_fork.c
new file mode 100644
index 0000000..24c4b16
--- /dev/null
+++ b/kernel/cgroup_fork.c
@@ -0,0 +1,180 @@
+/*
+ *  A cgroup implementation which limits the number of fork() calls.
+ *
+ *  This file is subject to the terms and conditions of the GNU General Public
+ *  License.  See the file COPYING in the main directory of the Linux
+ *  distribution for more details.
```

```
+ */
+
+#include <linux/cgroup.h>
+#include <linux/cgroup_fork.h>
+#include <linux/slab.h>
+
+struct cgroup_fork {
+ struct cgroup_subsys_state css;
+
+ /** protect the "remaining" attribute */
+ spinlock_t lock;
+
+ /**
+  * The remaining number of forks allowed.  -1 is the magic
+  * value for "unlimited".
+  */
+ int remaining;
+};
+
+/**
+ * Get the #cgrou_fork instance of the specified #cgroup.
+ */
+static inline struct cgroup_fork *
+cgroup_fork_group(struct cgroup *cgroup)
+{
+ return container_of(cgroup_subsys_state(cgroup, fork_subsys_id),
+       struct cgroup_fork, css);
+}
+
+/**
+ * Get the #cgroup_fork instance of the specified task.
+ */
+static inline struct cgroup_fork *
+cgroup_fork_task(struct task_struct *task)
+{
+ return container_of(task_subsys_state(current_task, fork_subsys_id),
+       struct cgroup_fork, css);
+}
+
+/**
+ * Get the #cgroup_fork instance of the current task.
+ */
+static inline struct cgroup_fork *
+cgroup_fork_current(void)
+{
+ return cgroup_fork_task(current_task);
+}
+
```

```
+static struct cgroup_subsys_state *
+cgroup_fork_create(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ struct cgroup_fork *t = kzalloc(sizeof(*t), GFP_KERNEL);
+ if (!t)
+  return ERR_PTR(-ENOMEM);
+
+ spin_lock_init(&t->lock);
+
+ t->remaining = -1;
+
+ return &t->css;
+}
+
+static void
+cgroup_fork_destroy(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ struct cgroup_fork *t = cgroup_fork_group(cgroup);
+
+ kfree(t);
+}
+
+static void
+cgroup_fork_fork(struct cgroup_subsys *ss, struct task_struct *task)
+{
+ struct cgroup_fork *t;
+
+ rcu_read_lock();
+
+ /* decrement the counters in the cgroup and all of its
+    ancestors (except for the root cgroup) */
+
+ t = cgroup_fork_current();
+ while (t->css.cgroup->parent != NULL) {
+  spin_lock_irq(&t->lock);
+  if (t->remaining > 0)
+   --t->remaining;
+  spin_unlock_irq(&t->lock);
+
+  t = cgroup_fork_group(t->css.cgroup->parent);
+ }
+
+ rcu_read_unlock();
+}
+
+static s64
+cgroup_fork_remaining_read(struct cgroup *cgroup, struct cftype *cft)
+{
```

```
+ struct cgroup_fork *t = cgroup_fork_group(cgroup);
+ int value;
+
+ spin_lock_irq(&t->lock);
+ value = t->remaining;
+ spin_unlock_irq(&t->lock);
+
+ return value;
+}
+
+static int
+cgroup_fork_remaining_write(struct cgroup *cgroup, struct cftype *cft,
+       s64 value)
+{
+ struct cgroup_fork *t = cgroup_fork_group(cgroup);
+
+ if (value < -1 || value > (1L << 30))
+  return -EINVAL;
+
+ spin_lock_irq(&t->lock);
+ t->remaining = (int)value;
+ spin_unlock_irq(&t->lock);
+
+ return 0;
+}
+
+static const struct cftype cgroup_fork_files[] =  {
+ {
+  .name = "remaining",
+  .read_s64 = cgroup_fork_remaining_read,
+  .write_s64 = cgroup_fork_remaining_write,
+ },
+};
+
+static int
+cgroup_fork_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ if (cgroup->parent == NULL)
+  /* cannot limit the root cgroup */
+  return 0;
+
+ return cgroup_add_files(cgroup, ss, cgroup_fork_files,
+     ARRAY_SIZE(cgroup_fork_files));
+}
+
+struct cgroup_subsys fork_subsys = {
+ .name = "fork",
+ .create = cgroup_fork_create,
```

```
+ .destroy = cgroup_fork_destroy,
+ .fork = cgroup_fork_fork,
+ .populate = cgroup_fork_populate,
+ .subsys_id = fork_subsys_id,
+};
+
+int
+cgroup_fork_pre_fork(void)
+{
+ struct cgroup_fork *t;
+ int err = 0;
+
+ rcu_read_lock();
+
+ t = cgroup_fork_current();
+ while (t->css.cgroup->parent != NULL && err == 0) {
+  spin_lock_irq(&t->lock);
+
+  if (t->remaining == 0)
+   err = -EPERM;
+
+  spin_unlock_irq(&t->lock);
+
+  t = cgroup_fork_group(t->css.cgroup->parent);
+ }
+
+ rcu_read_unlock();
+
+ return err;
+}
diff --git a/kernel/fork.c b/kernel/fork.c
index 25e4291..35836e6 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -32,6 +32,7 @@
 #include <linux/capability.h>
 #include <linux/cpu.h>
 #include <linux/cgroup.h>
+#include <linux/cgroup_fork.h>
 #include <linux/security.h>
 #include <linux/hugetlb.h>
 #include <linux/swap.h>
@@ -1024,6 +1025,10 @@ static struct task_struct *copy_process(unsigned long clone_flags,
     current->signal->flags & SIGNAL_UNKILLABLE)
   return ERR_PTR(-EINVAL);

+ retval = cgroup_fork_pre_fork();
+ if (retval)
```

```
+  goto fork_out;
+
  retval = security_task_create(clone_flags);
  if (retval)
   goto fork_out;
```

_____

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