Subject: Re: [PATCH 3/8] CGroup Files: Move the release_agent file to use typed handlers
Posted by serue on Tue, 24 Jun 2008 15:56:34 GMT
View Forum Message <> Reply to Message

Quoting menage@google.com (menage@google.com):
> Adds cgroup_release_agent_write() and cgroup_release_agent_show()
> methods to handle writing/reading the path to a cgroup hierarchy's
> release agent. As a result, cgroup_common_file_read() is now unnecessary.
>
> As part of the change, a previously-tolerated race in
> cgroup_release_agent() is avoided by copying the current
> release_agent_path prior to calling call_usermode_helper().
>
> Signed-off-by: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 2
> kernel/cgroup.c | 125 ++++++++++++++++++++++++++++++++----------------
> 2 files changed, 59 insertions(+), 68 deletions(-)
>
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -89,11 +89,7 @@ struct cgroupfs_root {
>   /* Hierarchy-specific flags */
>   unsigned long flags;
>
> - /* The path to use for release notifications. No locking
> - * between setting and use - so if userspace updates this
> - * while child cgroups exist, you could miss a
> - * notification. We ensure that it's always a valid
> - * NUL-terminated string */
> + /* The path to use for release notifications. */
>   char release_agent_path[PATH_MAX];
> };
>
> @@ -1329,6 +1325,45 @@ enum cgroup_filetype {
>   FILE_RELEASE_AGENT,
> };
>
> +/**
> + * cgroup_lock_live_group - take cgroup_mutex and check that cgrp is alive.
> + * @cgrp: the cgroup to be checked for liveness
> + *
> + * Returns true (with lock held) on success, or false (with no lock
> + * held) on failure.

> + */
> +int cgroup_lock_live_group(struct cgroup *cgrp)

Would seem more consistent to call the return type bool, but otherwise
this patch looks good.

Acked-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

> +{
> + mutex_lock(&cgroup_mutex);
> + if (cgroup_is_removed(cgrp)) {
> + mutex_unlock(&cgroup_mutex);
> + return false;
> + }
> + return true;
> +}
> +
> +static int cgroup_release_agent_write(struct cgroup *cgrp, struct cftype *cft,
> +   const char *buffer)
> +{
> + BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
> + if (!cgroup_lock_live_group(cgrp))
> + return -ENODEV;
> + strcpy(cgrp->root->release_agent_path, buffer);
> + mutex_unlock(&cgroup_mutex);
> + return 0;
> +}
> +
> +static int cgroup_release_agent_show(struct cgroup *cgrp, struct cftype *cft,
> +   struct seq_file *seq)
> +{
> + if (!cgroup_lock_live_group(cgrp))
> + return -ENODEV;
> + seq_puts(seq, cgrp->root->release_agent_path);
> + seq_putc(seq, '\n');
> + mutex_unlock(&cgroup_mutex);
> + return 0;
> +}
> +
> static ssize_t cgroup_write_X64(struct cgroup *cgrp, struct cftype *cft,
>   struct file *file,
>   const char __user *userbuf,
> @@ -1443,10 +1478,6 @@ static ssize_t cgroup_common_file_write(
>   else
>   clear_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);

```
>    break;
> - case FILE_RELEASE_AGENT:
> -  BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
> -  strcpy(cgrp->root->release_agent_path, buffer);
> -  break;
>   default:
>    retval = -EINVAL;
>    goto out2;
> @@ -1506,49 +1537,6 @@ static ssize_t cgroup_read_s64(struct cg
>   return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
>  }
>
> -static ssize_t cgroup_common_file_read(struct cgroup *cgrp,
> -       struct cftype *cft,
> -       struct file *file,
> -       char __user *buf,
> -       size_t nbytes, loff_t *ppos)
> -{
> - enum cgroup_filetype type = cft->private;
> - char *page;
> - ssize_t retval = 0;
> - char *s;
> -
> - if (!(page = (char *)__get_free_page(GFP_KERNEL)))
> -  return -ENOMEM;
> -
> - s = page;
> -
> - switch (type) {
> - case FILE_RELEASE_AGENT:
> - {
> -  struct cgroupfs_root *root;
> -  size_t n;
> -  mutex_lock(&cgroup_mutex);
> -  root = cgrp->root;
> -  n = strnlen(root->release_agent_path,
> -       sizeof(root->release_agent_path));
> -  n = min(n, (size_t) PAGE_SIZE);
> -  strncpy(s, root->release_agent_path, n);
> -  mutex_unlock(&cgroup_mutex);
> -  s += n;
> -  break;
> - }
> - default:
> -  retval = -EINVAL;
> -  goto out;
> - }
> - *s++ = '\n';
```

```
> -
> - retval = simple_read_from_buffer(buf, nbytes, ppos, page, s - page);
> -out:
> - free_page((unsigned long)page);
> - return retval;
> -}
> -
>  static ssize_t cgroup_file_read(struct file *file, char __user *buf,
>         size_t nbytes, loff_t *ppos)
>  {
> @@ -1606,6 +1594,7 @@ static int cgroup_seqfile_release(struct
>
>  static struct file_operations cgroup_seqfile_operations = {
>   .read = seq_read,
> + .write = cgroup_file_write,
>   .llseek = seq_lseek,
>   .release = cgroup_seqfile_release,
>  };
> @@ -2283,8 +2272,9 @@ static struct cftype files[] = {
>
>  static struct cftype cft_release_agent = {
>   .name = "release_agent",
> - .read = cgroup_common_file_read,
> - .write = cgroup_common_file_write,
> + .read_seq_string = cgroup_release_agent_show,
> + .write_string = cgroup_release_agent_write,
> + .max_write_len = PATH_MAX,
>   .private = FILE_RELEASE_AGENT,
>  };
>
> @@ -3113,27 +3103,24 @@ static void cgroup_release_agent(struct
>   while (!list_empty(&release_list)) {
>    char *argv[3], *envp[3];
>    int i;
> -  char *pathbuf;
> +  char *pathbuf = NULL, *agentbuf = NULL;
>    struct cgroup *cgrp = list_entry(release_list.next,
>           struct cgroup,
>           release_list);
>    list_del_init(&cgrp->release_list);
>    spin_unlock(&release_list_lock);
>    pathbuf = kmalloc(PAGE_SIZE, GFP_KERNEL);
> -  if (!pathbuf) {
> -   spin_lock(&release_list_lock);
> -   continue;
> -  }
> -
> -  if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0) {
```

> -   kfree(pathbuf);
> -   spin_lock(&release_list_lock);
> -   continue;
> - }
> + if (!pathbuf)
> +   goto continue_free;
> + if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0)
> +   goto continue_free;
> + agentbuf = kstrdup(cgrp->root->release_agent_path, GFP_KERNEL);
> + if (!agentbuf)
> +   goto continue_free;
>
>   i = 0;
> - argv[i++] = cgrp->root->release_agent_path;
> - argv[i++] = (char *)pathbuf;
> + argv[i++] = agentbuf;
> + argv[i++] = pathbuf;
>   argv[i] = NULL;
>
>   i = 0;
> @@ -3147,8 +3134,10 @@ static void cgroup_release_agent(struct
>   * be a slow process */
>   mutex_unlock(&cgroup_mutex);
>   call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
> - kfree(pathbuf);
>   mutex_lock(&cgroup_mutex);
> + continue_free:
> + kfree(pathbuf);
> + kfree(agentbuf);
>   spin_lock(&release_list_lock);
>   }
>   spin_unlock(&release_list_lock);
> Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> ===
> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -295,6 +295,8 @@ int cgroup_add_files(struct cgroup *cgrp
>
> int cgroup_is_removed(const struct cgroup *cgrp);
>
> +int cgroup_lock_live_group(struct cgroup *cgrp);
> +
> int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen);
>
> int cgroup_task_count(const struct cgroup *cgrp);
>
> --

_____

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