
Subject: Re: [PATCH 1/2] signal checkpoint: define /proc/pid/sig/
Posted by [Cedric Le Goater](#) on Mon, 11 Jun 2007 14:43:12 GMT
[View Forum Message](#) <> [Reply to Message](#)

Serge E. Hallyn wrote:

> As I mentioned earlier, I don't know what sort of approach we want
> to take to guide checkpoint and restart. I.e. do we want it to be
> a mostly userspace-orchestrated affair, or entirely done in the
> kernel using the freezer or some other mechanism in response to a
> single syscall or containerfs file write?

>
> If we wanted to do a lot of the work in userspace, here is a pair of
> patches to read and restore signal information. It's entirely unsafe
> wrt locking, bc i would assume that if we did in fact do c/r from
> userspace, we would have some way of entirely pulling the task off
> the runqueue while doing our thing...

>
> Anyway, this is purely to start discussion.

>
> thanks,
> -serge

>
>>From 30bbe322942e5ed86bad63861dad80595cd04063 Mon Sep 17 00:00:00 2001

> From: Serge E. Hallyn <serue@us.ibm.com>

> Date: Mon, 30 Apr 2007 16:22:44 -0400

> Subject: [PATCH 1/2] signal checkpoint: define /proc/pid/sig/

>
> Define /proc/<pid>/sig/ directory containing files to report
> on a process' signal info.

>
> Files defined:

> action: list signal action
> altstack: print sigaltstack location and size
> blocked: print blocked signal mask
> pending: print pending signals and siginfo
> shared_pending: print shared pending signals and siginfo
> waiters: list tasks wait4()ing on task PID.

some of these are already in /proc/<pid>/status and /proc/<pid>/stat

should we continue to use /proc ? or switch to some other mechanisms
like getnetlink (taskstats) to map kernel structures.

[...]

> +/*

> + * print a sigset_t to a buffer. Return # characters printed,

> + * not including the final ending '\0'.

```

> + */
> +static int print_sigset(char *buf, sigset_t *sig)
> +{
> + int i;
> +
> + for (i=0; i<_NSIG; i++) {
> + if (sigismember(sig, i))
> +   buf[i] = '1';
> + else
> +   buf[i] = '0';
> + }
> + buf[_NSIG] = '\0';
> +
> + return _NSIG;
> +}

```

you might want to use `render_sigset_t()` in `fs/proc/array.C`

```

> +static int print_sigset_alloc(char **bufp, sigset_t *sig)
> +{
> + char *buf;
> +
> + *bufp = buf = kmalloc(_NSIG+1, GFP_KERNEL);
> + if (!buf)
> +   return -ENOMEM;
> +
> + return print_sigset(buf, sig);
> +}
> +
> +static int print_wait_info(char **bufp, struct signal_struct *signal)
> +{
> + char *buf;
> + wait_queue_t *wait;
> +
> + *bufp = buf = kmalloc(PAGE_SIZE, GFP_KERNEL);
> + if (!buf)
> +   return -ENOMEM;
> +
> + spin_lock(&signal->wait_chldexit.lock);
> +
> + list_for_each_entry(wait, &signal->wait_chldexit.task_list, task_list) {
> +   struct task_struct *tsk = wait->private;
> +
> +   if (buf - *bufp + 50 > PAGE_SIZE) {
> +     spin_unlock(&signal->wait_chldexit.lock);
> +     kfree(buf);
> +     *bufp = NULL;
> +     return -ENOMEM;

```

```

> + }
> + WARN_ON(wait->func != default_wake_function);
> + buf += sprintf(buf, "%u %d\n", wait->flags, tsk->pid);
> + }
> +
> + spin_unlock(&signal->wait_chldexit.lock);
> +
> + return buf-*bufp;
> +}
> +
> +static int print_sigpending_alloc(char **bufp, struct sigpending *pending)
> +{
> + int allocated=0;
> + char *buf, *p;
> + struct sigqueue *q;
> + struct siginfo *info;
> +
> + allocated = PAGE_SIZE;
> + p = buf = kmalloc(allocated, GFP_KERNEL);
> + if (!buf)
> + return -ENOMEM;
> +
> + p += print_sigset(buf, &pending->signal);
> + p += sprintf(p, "\n");
> +
> + list_for_each_entry(q, &pending->list, list) {
> + info = &q->info;
> + if (p-buf+215 > allocated) {
> + int len=p-buf;
> + char *buf2;
> + allocated += PAGE_SIZE;
> + buf2 = kmalloc(allocated, GFP_KERNEL);
> + if (!buf2) {
> + kfree(buf);
> + return -ENOMEM;
> + }
> + memcpy(buf2, buf, allocated - PAGE_SIZE);
> + kfree(buf);
> + buf = buf2;
> + p = buf+len;
> + }
> +
> + p += sprintf(p, "sig %d: user %d flags %d",
> + info->si_signo, (int)q->user->uid, q->flags);
> + p += sprintf(p, " errno %d code %d\n",
> + info->si_errno, info->si_code);
> +
> + switch(info->si_signo) {

```

```

> + case SIGKILL:
> + p += sprintf(p, " spid %d suid %d\n",
> + info->_sifields._kill._pid,
> + info->_sifields._kill._uid);
> + break;
> + /* XXX skipping posix1b timers and signals for now */
> + case SIGCHLD:
> + p += sprintf(p, " pid %d uid %d status %d utime %lu stime %lu\n",
> + info->_sifields._sigchld._pid,
> + info->_sifields._sigchld._uid,
> + info->_sifields._sigchld._status,
> + info->_sifields._sigchld._utime,
> + info->_sifields._sigchld._stime);
> + break;
> + case SIGILL:
> + case SIGFPE:
> + case SIGSEGV:
> + case SIGBUS:
> + #ifdef __ARCH_SI_TRAPNO
> + p += sprintf(p, " addr %lu trapno %d\n",
> + (unsigned long)info->_sifields._sigfault._addr,
> + info->_sifields._sigfault._trapno);
> + #else
> + p += sprintf(p, " addr %lu\n",
> + (unsigned long)info->_sifields._sigfault._addr);
> + #endif
> + break;
> + case SIGPOLL:
> + p += sprintf(p, " band %ld fd %d\n",
> + (long)info->_sifields._sigpoll._band,
> + info->_sifields._sigpoll._fd);
> + break;
> + default:
> + p += sprintf(p, " Unsupported siginfo for signal %d\n",
> + info->si_signo);
> + break;
> + }
> + }
> + *bufp = buf;
> + return p-buf;
> +}

```

I think we are reaching the limit of /proc when we expose the pending siginfos.

```

> +static int print_sigaction_list(char **bufp, struct sighand_struct *sighand)
> +{
> + struct k_sigaction *action;
> + int maxlen;

```

```

> + int i;
> + char *buf;
> +
> + /* two unsigned longs (20 chars), one int (10 chars), a sigset_t, 3 spaces, plus a newline */
> + maxlen = 10 + 20*2 + _NSIG + 4;
> + /* and we have _NSIG of those entries, plus an ending \0 */
> + maxlen *= _NSIG+1;
> +
> + *bufp = buf = kmalloc(maxlen, GFP_KERNEL);
> + if (!buf)
> + return -ENOMEM;
> +
> + spin_lock(&sigband->siglock);
> + for (i=0; i<_NSIG; i++) {
> + action = &sigband->action[i];
> + buf += sprintf(buf, "%10d %20lu ", i, action->sa.sa_flags);
> + buf += print_sigset(buf, &action->sa.sa_mask);
> + buf += sprintf(buf, " %20lu\n", (unsigned long)action->sa.sa_handler);
> + BUG_ON(buf-*bufp > maxlen);
> + }
> + spin_unlock(&sigband->siglock);
> + return buf-*bufp;
> +}
> +
> +int task_read_procsig(struct task_struct *p, char *name, char **value)
> +{
> + int ret;
> +
> + if (current != p) {
> + ret = security_ptrace(current, p);
> + if (ret)
> + return ret;
> + }
> +
> + ret = -EINVAL;
> +
> + if (strcmp(name, "pending") == 0) {
> + /* not masking out blocked signals yet */
> + ret = print_sigpending_alloc(value, &p->pending);
> + } else if (strcmp(name, "blocked") == 0) {
> + /* not masking out blocked signals yet */
> + ret = print_sigset_alloc(value, &p->blocked);
> + } else if (strcmp(name, "shared_pending") == 0) {
> + ret = print_sigpending_alloc(value, &p->signal->shared_pending);
> + } else if (strcmp(name, "waiters") == 0) {
> + ret = print_wait_info(value, p->signal);
> + } else if (strcmp(name, "action") == 0) {
> + ret = print_sigaction_list(value, p->sigband);

```

```
> + } else if (strcmp(name, "altstack") == 0) {  
> + *value = kmalloc(40, GFP_KERNEL);  
> + ret = -ENOMEM;  
> + if (*value) {  
> + ret = sprintf(*value, "%lu %zd", p->sas_ss_sp, p->sas_ss_size);  
> + }  
> + }  
> + }  
> +  
> + return ret;  
> +}
```

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