

## Re: Someone help me understand this...?

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/current/2003-08/1814.html>

---

**From:** Robert Watson ([rwatson\\_at\\_freebsd.org](mailto:rwatson_at_freebsd.org))

**Date:** 08/30/03

Date: Sat, 30 Aug 2003 12:23:35 -0400 (EDT)

To: Jilles Tjoelker <[jilles@stack.nl](mailto:jilles@stack.nl)>

On Sat, 30 Aug 2003, Jilles Tjoelker wrote:

> *On Thu, Aug 28, 2003 at 11:34:09AM -0400, Robert Watson wrote:*  
> > > *Clearly, unbreaking applications like Diablo by default is desirable. At*  
> > > *least OpenBSD has similar protections to these turned on by default, and*  
> > > *possibly other systems as well. As 5.x sees more broad use, we may well*  
> > > *bump into other cases where applications have similar behavior: they rely*  
> > > *on no special protections once they've given up privilege. I wonder if*  
> > > *Diablo can run unmodified on OpenBSD; it could be they don't include*  
> > > *SIGALRM on the list of "protect against" signals, or it could be that they*  
> > > *modify Diablo for their environment to use an alternative signaling*  
> > > *mechanism. Another alternative to this patch would simply be to add*  
> > > *SIGARLM to the list of acceptable signals to deliver in the*  
> > > *privilege-change case.*  
>  
> *OpenBSD does not consider a process 'tainted' if it changes credentials*  
> *while running. From the issetugid(2) manpage:*  
>  
> *The status of issetugid() is only affected by execve().*

In OpenBSD, two flags are used to represent the credential change notion: P\_SUGIDEXEC, and P\_SUGID. issetugid() checks the first of these, but signal delivery checks P\_SUGID. P\_SUGIDEXEC is set during execve(). In FreeBSD, we have a combined notion used by both, since the same protections generally apply. You can find a comment comparing our use of P\_SUGID to the OpenBSD approach in our issetugid() implementation:

```
/*  
 * Note: OpenBSD sets a P_SUGIDEXEC flag set at execve() time,  
 * we use P_SUGID because we consider changing the owners as  
 * "tainting" as well.  
 * This is significant for procs that start as root and "become"  
 * a user without an exec - programs cannot know *everything*  
 * that libc *might* have put in their data segment.  
 */
```

## freebsd-current: Re: Someone help me understand this...?

Regarding specific signals: inspection of the OpenBSD implementation reveals that the following signals are permitted in the P\_SUGID case, assuming a reasonable credential match:

- case 0:
- case SIGKILL:
- case SIGINT:
- case SIGTERM:
- case SIGALRM:
- case SIGSTOP:
- case SIGTTIN:
- case SIGTTOU:
- case SIGTSTP:
- case SIGHUP:
- case SIGUSR1:
- case SIGUSR2:

In FreeBSD, we permit:

- case 0:
- case SIGKILL:
- case SIGINT:
- case SIGTERM:
- case SIGSTOP:
- case SIGTTIN:
- case SIGTTOU:
- case SIGTSTP:
- case SIGHUP:
- case SIGUSR1:
- case SIGUSR2:

So they permit SIGALRM in addition to the signals we support. In light of this thread, I think it would be reasonable to add SIGALRM to our list as well.

> > *In most cases, fail-stop is a reasonable behavior for unexpected security*  
> > *behavior from the system, but ignore is likely to shoot you later. :-)* I  
> > *tend to wrap even kill() calls as uid 0 in an assertion check, just to be*  
> > *on the safe side. If nothing else, it helps detect the case where the*  
> > *other process has died, and you're using a stale pid. It's particular*  
> > *useful if the other process has died, the pid has been reused, and it's*  
> > *now owned by another user, which is a real-world case where kill() as a*  
> > *non-0 uid can fail even when you're sure it can't :-).*  
>  
> *This can be avoided by careful programming: do not use SA\_NOCLDWAIT and*  
> *don't pass pids to kill() when they have been returned by wait() or*  
> *similar functions. If the process has terminated in between, it's a*  
> *zombie. In that case, FreeBSD probably returns ESRCH but SUSv3 mandates*  
> *returning success (but performing no action).*

freebsd-current: Re: Someone help me understand this...?

There's still a race possible here, it just becomes more narrow with conservative programming. And in the classic use of pids for signalling (/var/run/foo.pid, or kill -9 pid), these approaches won't help. The only way to close this sort of race is to have a notion of a unique process identifier that lasts beyond the lifetime of the process itself -- i.e., the ability to return EMYSINCERESTREGRESTS if you try to signal a process after it has died, and have a guarantee that the handle won't be reused.

Robert N M Watson FreeBSD Core Team, TrustedBSD Projects  
robert@fledge.watson.org Network Associates Laboratories

---

freebsd-current@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-current>

To unsubscribe, send any mail to "freebsd-current-unsubscribe@freebsd.org"