Subject: Re: [PATCH] fdset's leakage
Posted by Vadim Lobanov on Tue, 11 Jul 2006 16:13:38 GMT
View Forum Message <> Reply to Message

On Tue, 11 Jul 2006, Kirill Korotaev wrote:

> Andrew,
>
> >>Another patch from Alexey Kuznetsov fixing memory leak in alloc_fdtable().
> >>
> >>[PATCH] fdset's leakage
> >>
> >>When found, it is obvious. nfds calculated when allocating fdsets
> >>is rewritten by calculation of size of fdtable, and when we are
> >>unlucky, we try to free fdsets of wrong size.
> >>
> >>Found due to OpenVZ resource management (User Beancounters).
> >>
> >>Signed-Off-By: Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>
> >>Signed-Off-By: Kirill Korotaev <dev@openvz.org>
> >>
> >>
> >>diff -urp linux-2.6-orig/fs/file.c linux-2.6/fs/file.c
> >>--- linux-2.6-orig/fs/file.c 2006-07-10 12:10:51.000000000 +0400
> >>+++ linux-2.6/fs/file.c 2006-07-10 14:47:01.000000000 +0400
> >>@@ -277,11 +277,13 @@ static struct fdtable *alloc_fdtable(int
> >> } while (nfds <= nr);
> >> new_fds = alloc_fd_array(nfds);
> >> if (!new_fds)
> >>- goto out;
> >>+ goto out2;
> >> fdt->fd = new_fds;
> >> fdt->max_fds = nfds;
> >> fdt->free_files = NULL;
> >> return fdt;
> >>+out2:
> >>+ nfds = fdt->max_fdset;
> >> out:
> >>   if (new_openset)
> >>   free_fdset(new_openset, nfds);
> >
> >
> > OK, that was a simple fix. And if we need this fix backported to 2.6.17.x
> > then it'd be best to go with the simple fix.
> >
> > And I think we do need to backport this to 2.6.17.x because NR_OPEN can be
> > really big, and vmalloc() is not immortal.
> >

> > But the code in there is really sick. In all cases we do:
> >
> > free_fdset(foo->open_fds, foo->max_fdset);
> > free_fdset(foo->close_on_exec, foo->max_fdset);
> >
> > How much neater and more reliable would it be to do:
> >
> > free_fdsets(foo);
> >
> > ?
> agree. should I prepare a patch?
>
> > Also,
> >
> > nfds = NR_OPEN_DEFAULT;
> > /*
> >   * Expand to the max in easy steps, and keep expanding it until
> >   * we have enough for the requested fd array size.
> >   */
> > do {
> > #if NR_OPEN_DEFAULT < 256
> >   if (nfds < 256)
> >   nfds = 256;
> >   else
> > #endif
> >   if (nfds < (PAGE_SIZE / sizeof(struct file *)))
> >   nfds = PAGE_SIZE / sizeof(struct file *);
> >   else {
> >   nfds = nfds * 2;
> >   if (nfds > NR_OPEN)
> >   nfds = NR_OPEN;
> >   }
> > } while (nfds <= nr);
> >
> >
> > That's going to take a long time to compute if nr > NR_OPEN. I just fixed
> > a similar infinite loop in this function. Methinks this
> >
> > nfds = max(NR_OPEN_DEFAULT, 256);
> > nfds = max(nfds, PAGE_SIZE/sizeof(struct file *));
> > nfds = max(nfds, round_up_pow_of_two(nr + 1));
> > nfds = min(nfds, NR_OPEN);
> >
> > is clearer and less buggy. I _think_ it's also equivalent (as long as
> > NR_OPEN>256). But please check my logic.
> Yeah, I also noticed these nasty loops but was too lazy to bother :)
> Too much crap for my nerves :)
>

> Your logic looks fine for me. Do we have already round_up_pow_of_two() function or
> should we create it as something like:
> unsinged long round_up_pow_of_two(unsigned long x)
> {
> unsigned long res = 1 << BITS_PER_LONG;

You'll get a zero here. Should be 1 << (BITS_PER_LONG - 1).

> while (res > x)
> res >>= 1;
> }
> return res << 1;
> }
>
> or maybe using:
> n = find_first_bit(x);
> return res = 1 << n;
> (though it depends on endianness IMHO)
> ?
>
> Thanks,
> Kirill

-- Vadim Lobanov