

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-04/msg00673.html>

- *From:* Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>
 - *Date:* Wed, 02 Apr 2008 13:47:47 +0200
-

On Wed, 2008-04-02 at 13:42 +0200, Jens Axboe wrote:

On Wed, Apr 02 2008, Peter Zijlstra wrote:

On Wed, 2008-04-02 at 13:32 +0200, Jens Axboe wrote:

On Wed, Apr 02 2008, Peter Zijlstra wrote:

On Wed, 2008-04-02 at 13:14 +0200, Jens Axboe wrote:

On Wed, Apr 02 2008, Peter Zijlstra wrote:

On Wed,
2008-04-02
at 13:07
+0200, Jens
Axboe
wrote:

On
Wed,
Apr
02
2008,
Pekka
Enberg
wrote:

Hi
Paul,

On
Wed,
Apr
2,
2008

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

at
1:55
PM,
Paul
E.
McKenney
<paulmck@xxxxxxxxxxxxxxxxxxxx>
wrote:

I
will
check
this
when
I
get
back
to
some
bandwidth
--
but
in
the
meantime,
does
kmemcheck
special-case
SLAB_DESTROY_BY_RCU?
It
is
legal
to
access
newly-freed
items
in
that
case,
as
long
as
you
did
rcu_read_lock()
before
gaining
a
reference
to
them

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

and
don't
hold
the
reference
past
the
matching
rcu_read_unlock().

No,
kmemcheck
is
work
in
progress
and
does
not
know
about
SLAB_DESTROY_BY_RCU
yet.
The
reason
I
asked
Vegard
to
post
the
warning
was
because
Peter,
Vegard,
and
myself
identified
this
particular
warning
as
a
real
problem.
But
yeah,
kmemcheck
can

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

cause
false
positives
for
RCU
for
now.

Makes
sense,
and
to
me
Pauls
analysis
of
the
code
looks
totally
correct
–
there's
no
bug
there,
at
least
related
to
hlist
traversal
and
kmem_cache_free(),
since
we
are
under
rcu_read_lock()
and
thus
hold
off
the
grace
for
freeing.

but what

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

holds off
the slab
allocator
re-issuing
that same
object and
someone
else writing
other stuff
into it?

Nothing, that's how rcu
destry works here. But for
the validation to be
WRONG
radix_tree_lookup(...,
old_key) must return cic for
new_key, not
NULL.

A B C

```
cfq_cic_lookup(cfqd_1, ioc)
```

```
rcu_read_lock()  
cic = radix_tree_lookup(, cfqd_q);
```

```
cfq_cic_free()
```

```
cfq_cic_link(cfqd_2, ioc,)
```

```
rcu_read_unlock()
```

and now we have that:

```
cic->key == cfqd_2
```

I'm not seeing anything stopping this from
happening.

I don't follow your A-B-C here, what do they refer to?

A does a radix_tree_lookup() of cfqd_1 (darn typos)

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

Re: kmemcheck caught read from freed memory (cfq_free_io_context)

B does a kfree of the same cic found by A
C does an alloc and gets the same cic as freed by B and inserts it
in a different location.

So that when we return to A, cic->key == cfqd_2 even though we did a
lookup for cfqd_1.

That I follow, my question was if A, B, and C refer to different
processes but with a shared io context? I'm assuming that is correct...

Ah, yeah, whatever is needed to make this race happen :-)

And it does look buggy. It looks my assumption of what slab rcu destroy
did is WRONG, it should be replaced by a manual call_rcu() freeing
instead.

Yeah, SLAB_DESTROY_BY_RCU should have a `_HUGE_` comment explaining it,
I'm sure this is not the first (nor the last) time people get that
wrong.

This would be one of those things that score very low on Rusty's API
list.

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>