Subject: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by xemul on Wed, 18 Apr 2007 08:12:18 GMT
View Forum Message <> Reply to Message

Sorry, I forgot to put netdev and David in Cc when I first sent it.

There is a race between netlink_dump_start() and netlink_release()
that can lead to the situation when a netlink socket with non-zero
callback is freed.

Here it is:

```
CPU1:                       CPU2
netlink_release():          netlink_dump_start():

                        sk = netlink_lookup(); /* OK */

netlink_remove();

spin_lock(&nlk->cb_lock);
if (nlk->cb) { /* false */
  ...
}
spin_unlock(&nlk->cb_lock);

                        spin_lock(&nlk->cb_lock);
                        if (nlk->cb) { /* false */
                                ...
                        }
                        nlk->cb = cb;
                        spin_unlock(&nlk->cb_lock);
                        ...
sock_orphan(sk);
/*
 * proceed with releasing
 * the socket
 */
```

The proposal it to make sock_orphan before detaching the callback
in netlink_release() and to check for the sock to be SOCK_DEAD in
netlink_dump_start() before setting a new callback.

Signed-off-by: Denis Lunev <den@openvz.org>
Signed-off-by: Kirill Korotaev <dev@openvz.org>
Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

```
--- a/net/netlink/af_netlink.c 2004-10-25 12:12:23.000000000 +0400
+++ b/net/netlink/af_netlink.c 2004-10-28 16:26:12.000000000 +0400
@@ -255,6 +255,7 @@ static int netlink_release(struct socket
   return 0;

  netlink_remove(sk);
+ sock_orphan(sk);
  nlk = nlk_sk(sk);

  spin_lock(&nlk->cb_lock);
@@ -269,7 +270,6 @@ static int netlink_release(struct socket
  /* OK. Socket is unlinked, and, therefore,
     no new packets will arrive */

- sock_orphan(sk);
  sock->sk = NULL;
  wake_up_interruptible_all(&nlk->wait);

@@ -942,9 +942,9 @@ int netlink_dump_start(struct sock *ssk,
   return -ECONNREFUSED;
  }
  nlk = nlk_sk(sk);
- /* A dump is in progress... */
+ /* A dump or destruction is in progress... */
  spin_lock(&nlk->cb_lock);
- if (nlk->cb) {
+ if (nlk->cb || sock_flag(sk, SOCK_DEAD)) {
   spin_unlock(&nlk->cb_lock);
   netlink_destroy_callback(cb);
   sock_put(sk);
```

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 08:17:07 GMT

View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
> Sorry, I forgot to put netdev and David in Cc when I first sent it.
>
> There is a race between netlink_dump_start() and netlink_release()
> that can lead to the situation when a netlink socket with non-zero
> callback is freed.

Out of curiosity, why not to fix a netlink_dump_start() to remove
callback in error path, since in 'no-error' path it removes it in
netlink_dump().

And, btw, can release method be called while socket is being used, I
thought about proper reference counters should prevent this, but not
100% sure with RCU dereferencing of the descriptor.

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Patrick McHardy on Wed, 18 Apr 2007 08:26:31 GMT
View Forum Message <> Reply to Message

Evgeniy Polyakov wrote:
> On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
>
>>Sorry, I forgot to put netdev and David in Cc when I first sent it.
>>
>>There is a race between netlink_dump_start() and netlink_release()
>>that can lead to the situation when a netlink socket with non-zero
>>callback is freed.
>
>
> Out of curiosity, why not to fix a netlink_dump_start() to remove
> callback in error path, since in 'no-error' path it removes it in
> netlink_dump().

It already does (netlink_destroy_callback), but that doesn't help
with this race though since without this patch we don't enter the
error path.

> And, btw, can release method be called while socket is being used, I
> thought about proper reference counters should prevent this, but not
> 100% sure with RCU dereferencing of the descriptor.

The problem is asynchronous processing of the dump request in the
context of a different process. Process requests a dump, message
is queued and process returns from sendmsg since some other process
is already processing the queue. Then the process closes the socket,
resulting in netlink_release being called. When the dump request
is finally processed the race Pavel described might happen. This
can only happen for netlink families that use mutex_try_lock for
queue processing of course.

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket

Posted by xemul on Wed, 18 Apr 2007 08:27:56 GMT
View Forum Message <> Reply to Message

Evgeniy Polyakov wrote:
> On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
>> Sorry, I forgot to put netdev and David in Cc when I first sent it.
>>
>> There is a race between netlink_dump_start() and netlink_release()
>> that can lead to the situation when a netlink socket with non-zero
>> callback is freed.
>
> Out of curiosity, why not to fix a netlink_dump_start() to remove
> callback in error path, since in 'no-error' path it removes it in

Error path is not relevant here. The problem is that we
keep a calback on a socket that is about to be freed.

> netlink_dump().
>
> And, btw, can release method be called while socket is being used, I
> thought about proper reference counters should prevent this, but not
> 100% sure with RCU dereferencing of the descriptor.
>

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 08:42:07 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 10:26:31AM +0200, Patrick McHardy (kaber@trash.net) wrote:
> Evgeniy Polyakov wrote:
> > On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
> >
> >>Sorry, I forgot to put netdev and David in Cc when I first sent it.
> >>
> >>There is a race between netlink_dump_start() and netlink_release()
> >>that can lead to the situation when a netlink socket with non-zero
> >>callback is freed.
> >
> >
> > Out of curiosity, why not to fix a netlink_dump_start() to remove
> > callback in error path, since in 'no-error' path it removes it in
> > netlink_dump().
>
>
> It already does (netlink_destroy_callback), but that doesn't help
> with this race though since without this patch we don't enter the
> error path.

I thought that with releasing a socket, which will have a callback
attached only results in a leak of the callback? In that case we can
just free it in dump() just like it is done in no-error path already.
Or do I miss something additional?

> > And, btw, can release method be called while socket is being used, I
> > thought about proper reference counters should prevent this, but not
> > 100% sure with RCU dereferencing of the descriptor.
>
>
> The problem is asynchronous processing of the dump request in the
> context of a different process. Process requests a dump, message
> is queued and process returns from sendmsg since some other process
> is already processing the queue. Then the process closes the socket,
> resulting in netlink_release being called. When the dump request
> is finally processed the race Pavel described might happen. This
> can only happen for netlink families that use mutex_try_lock for
> queue processing of course.

Doesn't it called from ->sk_data_ready() which is synchronous with
respect to sendmsg, not sure about conntrack though, but it looks so?

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 08:44:16 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 12:32:40PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
> Evgeniy Polyakov wrote:
> > On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
> >> Sorry, I forgot to put netdev and David in Cc when I first sent it.
> >>
> >> There is a race between netlink_dump_start() and netlink_release()
> >> that can lead to the situation when a netlink socket with non-zero
> >> callback is freed.
> >
> > Out of curiosity, why not to fix a netlink_dump_start() to remove
> > callback in error path, since in 'no-error' path it removes it in
>
> Error path is not relevant here. The problem is that we
> keep a calback on a socket that is about to be freed.

Yes, you are right, that it will not be freed in netlink_release(),
but it will be freed in netlink_dump() after it is processed (in no-error

path only though).

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Patrick McHardy on Wed, 18 Apr 2007 08:50:42 GMT
View Forum Message <> Reply to Message

Evgeniy Polyakov wrote:
> On Wed, Apr 18, 2007 at 10:26:31AM +0200, Patrick McHardy (kaber@trash.net) wrote:
>
>>>Out of curiosity, why not to fix a netlink_dump_start() to remove
>>>callback in error path, since in 'no-error' path it removes it in
>>>netlink_dump().
>>
>>
>>It already does (netlink_destroy_callback), but that doesn't help
>>with this race though since without this patch we don't enter the
>>error path.
>
>
> I thought that with releasing a socket, which will have a callback
> attached only results in a leak of the callback? In that case we can
> just free it in dump() just like it is done in no-error path already.
> Or do I miss something additional?

That would only work if there is nothing to dump (cb->dump returns 0).
Otherwise it is not freed.

>>The problem is asynchronous processing of the dump request in the
>>context of a different process. Process requests a dump, message
>>is queued and process returns from sendmsg since some other process
>>is already processing the queue. Then the process closes the socket,
>>resulting in netlink_release being called. When the dump request
>>is finally processed the race Pavel described might happen. This
>>can only happen for netlink families that use mutex_try_lock for
>>queue processing of course.
>
>
> Doesn't it called from ->sk_data_ready() which is synchronous with
> respect to sendmsg, not sure about conntrack though, but it looks so?

Yes, but for kernel sockets we end up calling the input function,
which when mutex_trylock is used returns immediately when some

other process is already processing the queue, so the requesting
process might close the socket before the request is processed.

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by xemul on Wed, 18 Apr 2007 08:59:20 GMT
View Forum Message <> Reply to Message

Evgeniy Polyakov wrote:
> On Wed, Apr 18, 2007 at 12:32:40PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
>> Evgeniy Polyakov wrote:
>>> On Wed, Apr 18, 2007 at 12:16:18PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
>>>> Sorry, I forgot to put netdev and David in Cc when I first sent it.
>>>>
>>>> There is a race between netlink_dump_start() and netlink_release()
>>>> that can lead to the situation when a netlink socket with non-zero
>>>> callback is freed.
>>> Out of curiosity, why not to fix a netlink_dump_start() to remove
>>> callback in error path, since in 'no-error' path it removes it in
>> Error path is not relevant here. The problem is that we
>> keep a calback on a socket that is about to be freed.
>
> Yes, you are right, that it will not be freed in netlink_release(),
> but it will be freed in netlink_dump() after it is processed (in no-error
> path only though).
>

But error path will leak it. On success path we would have
a leaked packet in sk_write_queue, since we did't see it in
skb_queue_purge() while doing netlink_release().

Of course we can place the struts in code to handle the case
when we have a released socket with the attached callback, but
it is more correct (IMHO) not to allow to attach the callbacks
to dead sockets.

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 09:07:20 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 10:50:42AM +0200, Patrick McHardy (kaber@trash.net) wrote:
> >>It already does (netlink_destroy_callback), but that doesn't help
> >>with this race though since without this patch we don't enter the
> >>error path.
> >
> > I thought that with releasing a socket, which will have a callback

> > attached only results in a leak of the callback? In that case we can
> > just free it in dump() just like it is done in no-error path already.
> > Or do I miss something additional?
>
> That would only work if there is nothing to dump (cb->dump returns 0).
> Otherwise it is not freed.

That is what I referred to as error path. Btw, with positive return
value we end up in subsequent call to input which will free callback
under lock as expected.

I do not object against the patch, just want to make a clear vision about
dumps - if callback is allocated to be used in dump only, then we could
just free it there without passing to next round.

> >>The problem is asynchronous processing of the dump request in the
> >>context of a different process. Process requests a dump, message
> >>is queued and process returns from sendmsg since some other process
> >>is already processing the queue. Then the process closes the socket,
> >>resulting in netlink_release being called. When the dump request
> >>is finally processed the race Pavel described might happen. This
> >>can only happen for netlink families that use mutex_try_lock for
> >>queue processing of course.
> >
> >
> > Doesn't it called from ->sk_data_ready() which is synchronous with
> > respect to sendmsg, not sure about conntrack though, but it looks so?
>
>
> Yes, but for kernel sockets we end up calling the input function,
> which when mutex_trylock is used returns immediately when some
> other process is already processing the queue, so the requesting
> process might close the socket before the request is processed.

So far it is only netfilter and gennetlink, we would see huge dump
from netlink_sock_destruct.
Anyway, that is possible situation, thanks for clearing this up.

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 09:14:18 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 01:03:56PM +0400, Pavel Emelianov (xemul@sw.ru) wrote:
> > Yes, you are right, that it will not be freed in netlink_release(),

> > but it will be freed in netlink_dump() after it is processed (in no-error
> > path only though).
> >
>
> But error path will leak it. On success path we would have
> a leaked packet in sk_write_queue, since we did't see it in
> skb_queue_purge() while doing netlink_release().
>
> Of course we can place the struts in code to handle the case
> when we have a released socket with the attached callback, but
> it is more correct (IMHO) not to allow to attach the callbacks
> to dead sockets.

That is why I've asked why such approach is used but not freeing
callback in errror (well, no-dump name is better to describe that path)
path, and more generally, why callback is attached, but not freed in the
function, but instead is freed next time dump started.

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Patrick McHardy on Wed, 18 Apr 2007 09:16:50 GMT
View Forum Message <> Reply to Message

Evgeniy Polyakov wrote:
> On Wed, Apr 18, 2007 at 10:50:42AM +0200, Patrick McHardy (kaber@trash.net) wrote:
>
>>>I thought that with releasing a socket, which will have a callback
>>>attached only results in a leak of the callback? In that case we can
>>>just free it in dump() just like it is done in no-error path already.
>>>Or do I miss something additional?
>>
>>That would only work if there is nothing to dump (cb->dump returns 0).
>>Otherwise it is not freed.
>
>
> That is what I referred to as error path. Btw, with positive return
> value we end up in subsequent call to input which will free callback
> under lock as expected.

No, nothing is going to call netlink_dump after the initial call since
the socket is gone.

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Evgeniy Polyakov on Wed, 18 Apr 2007 09:29:03 GMT
View Forum Message <> Reply to Message

On Wed, Apr 18, 2007 at 11:16:50AM +0200, Patrick McHardy (kaber@trash.net) wrote:
> > That is what I referred to as error path. Btw, with positive return
> > value we end up in subsequent call to input which will free callback
> > under lock as expected.
>
>
> No, nothing is going to call netlink_dump after the initial call since
> the socket is gone.

Argh, userspace socket's sk_data_rady() if dump returned positive value.
So, callback is not freed to allow to put several pages before
NLMSG_DONE via single dump?

--
Evgeniy Polyakov

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by davem on Thu, 19 Apr 2007 00:06:36 GMT
View Forum Message <> Reply to Message

From: Pavel Emelianov <xemul@sw.ru>
Date: Wed, 18 Apr 2007 12:16:18 +0400

> The proposal it to make sock_orphan before detaching the callback
> in netlink_release() and to check for the sock to be SOCK_DEAD in
> netlink_dump_start() before setting a new callback.

As discussed in this thread there might be other ways to a
approach this, but this fix is good for now.

Patch applied, thank you.

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Herbert Xu on Thu, 19 Apr 2007 02:13:51 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> wrote:
>
> As discussed in this thread there might be other ways to a
> approach this, but this fix is good for now.
>
> Patch applied, thank you.

Actually I was going to suggest something like this:

[NETLINK]: Kill CB only when socket is unused

Since we can still receive packets until all references to the
socket are gone, we don't need to kill the CB until that happens.
This also aligns ourselves with the receive queue purging which
happens at that point.

Original patch by Pavel Emelianov who noticed this race condition.

Signed-off-by: Herbert Xu <herbert@gondor.apana.org.au>

Cheers,
--
Visit Openswan at http://www.openswan.org/
Email: Herbert Xu ~{PmV>HI~} <herbert@gondor.apana.org.au>
Home Page: http://gondor.apana.org.au/~herbert/
PGP Key: http://gondor.apana.org.au/~herbert/pubkey.txt
--

```
diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index 0be19b7..914884c 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -139,6 +139,15 @@ static struct hlist_head *nl_pid_hashfn(struct nl_pid_hash *hash, u32
pid)

 static void netlink_sock_destruct(struct sock *sk)
 {
+ struct netlink_sock *nlk = nlk_sk(sk);
+
+ WARN_ON(mutex_is_locked(nlk_sk(sk)->cb_mutex));
+ if (nlk->cb) {
+  if (nlk->cb->done)
+   nlk->cb->done(nlk->cb);
+  netlink_destroy_callback(nlk->cb);
+ }
+
  skb_queue_purge(&sk->sk_receive_queue);

  if (!sock_flag(sk, SOCK_DEAD)) {
@@ -147,7 +156,6 @@ static void netlink_sock_destruct(struct sock *sk)
  }
  BUG_TRAP(!atomic_read(&sk->sk_rmem_alloc));
  BUG_TRAP(!atomic_read(&sk->sk_wmem_alloc));
- BUG_TRAP(!nlk_sk(sk)->cb);
  BUG_TRAP(!nlk_sk(sk)->groups);
```

```
 }

@@ -450,17 +458,7 @@ static int netlink_release(struct socket *sock)
  netlink_remove(sk);
  nlk = nlk_sk(sk);

- mutex_lock(nlk->cb_mutex);
- if (nlk->cb) {
-  if (nlk->cb->done)
-   nlk->cb->done(nlk->cb);
-  netlink_destroy_callback(nlk->cb);
-  nlk->cb = NULL;
- }
- mutex_unlock(nlk->cb_mutex);
-
- /* OK. Socket is unlinked, and, therefore,
-    no new packets will arrive */
+ /* OK. Socket is unlinked. */

  sock_orphan(sk);
  sock->sk = NULL;
```

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by davem on Sun, 29 Apr 2007 06:18:49 GMT
View Forum Message <> Reply to Message

From: Herbert Xu <herbert@gondor.apana.org.au>
Date: Thu, 19 Apr 2007 12:13:51 +1000

> [NETLINK]: Kill CB only when socket is unused
>
> Since we can still receive packets until all references to the
> socket are gone, we don't need to kill the CB until that happens.
> This also aligns ourselves with the receive queue purging which
> happens at that point.
>
> Original patch by Pavel Emelianov who noticed this race condition.
>
> Signed-off-by: Herbert Xu <herbert@gondor.apana.org.au>

Herbert, could you refresh this refinement to the current
tree?

Thanks a lot!

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by Herbert Xu on Wed, 02 May 2007 04:12:22 GMT
View Forum Message <> Reply to Message

On Sat, Apr 28, 2007 at 11:18:49PM -0700, David Miller wrote:
>
> Herbert, could you refresh this refinement to the current
> tree?

Dave, thanks for reminding me. Here it is.

[NETLINK]: Kill CB only when socket is unused

Since we can still receive packets until all references to the
socket are gone, we don't need to kill the CB until that happens.
This also aligns ourselves with the receive queue purging which
happens at that point.

Signed-off-by: Herbert Xu <herbert@gondor.apana.org.au>

Cheers,
--
Visit Openswan at http://www.openswan.org/
Email: Herbert Xu ~{PmV>HI~} <herbert@gondor.apana.org.au>
Home Page: http://gondor.apana.org.au/~herbert/
PGP Key: http://gondor.apana.org.au/~herbert/pubkey.txt
--
diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index 42d2fb9..7fc6b4d 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -140,6 +140,15 @@ static struct hlist_head *nl_pid_hashfn(struct nl_pid_hash *hash, u32 pid)

 static void netlink_sock_destruct(struct sock *sk)
 {
+ struct netlink_sock *nlk = nlk_sk(sk);
+
+ BUG_ON(mutex_is_locked(nlk_sk(sk)->cb_mutex));
+ if (nlk->cb) {
+ if (nlk->cb->done)
+   nlk->cb->done(nlk->cb);
+ netlink_destroy_callback(nlk->cb);
+ }
+
  skb_queue_purge(&sk->sk_receive_queue);

  if (!sock_flag(sk, SOCK_DEAD)) {
@@ -148,7 +157,6 @@ static void netlink_sock_destruct(struct sock *sk)

```
  }
  BUG_TRAP(!atomic_read(&sk->sk_rmem_alloc));
  BUG_TRAP(!atomic_read(&sk->sk_wmem_alloc));
- BUG_TRAP(!nlk_sk(sk)->cb);
  BUG_TRAP(!nlk_sk(sk)->groups);
 }

@@ -456,17 +464,10 @@ static int netlink_release(struct socket *sock)
  sock_orphan(sk);
  nlk = nlk_sk(sk);

- mutex_lock(nlk->cb_mutex);
- if (nlk->cb) {
-  if (nlk->cb->done)
-   nlk->cb->done(nlk->cb);
-  netlink_destroy_callback(nlk->cb);
-  nlk->cb = NULL;
- }
- mutex_unlock(nlk->cb_mutex);
-
- /* OK. Socket is unlinked, and, therefore,
-    no new packets will arrive */
+ /*
+  * OK. Socket is unlinked, any packets that arrive now
+  * will be purged.
+  */

  sock->sk = NULL;
  wake_up_interruptible_all(&nlk->wait);
@@ -1426,9 +1427,9 @@ int netlink_dump_start(struct sock *ssk, struct sk_buff *skb,
   return -ECONNREFUSED;
  }
  nlk = nlk_sk(sk);
- /* A dump or destruction is in progress... */
+ /* A dump is in progress... */
  mutex_lock(nlk->cb_mutex);
- if (nlk->cb || sock_flag(sk, SOCK_DEAD)) {
+ if (nlk->cb) {
   mutex_unlock(nlk->cb_mutex);
   netlink_destroy_callback(cb);
   sock_put(sk);
```

Subject: Re: [NETLINK] Don't attach callback to a going-away netlink socket
Posted by davem on Thu, 03 May 2007 10:17:27 GMT
View Forum Message <> Reply to Message

From: Herbert Xu <herbert@gondor.apana.org.au>

Date: Wed, 2 May 2007 14:12:22 +1000

> Dave, thanks for reminding me. Here it is.
>
> [NETLINK]: Kill CB only when socket is unused
>
> Since we can still receive packets until all references to the
> socket are gone, we don't need to kill the CB until that happens.
> This also aligns ourselves with the receive queue purging which
> happens at that point.
>
> Signed-off-by: Herbert Xu <herbert@gondor.apana.org.au>

Applied, thanks Herbert.
