Subject: [PATCH (resubmit)] Fix inet_diag.ko register vs rcv race
Posted by Pavel Emelianov on Thu, 29 Nov 2007 13:01:25 GMT
View Forum Message <> Reply to Message

The following race is possible when one cpu unregisters the handler
while other one is trying to receive a message and call this one:

```
CPU1:                                   CPU2:
inet_diag_rcv()                           inet_diag_unregister()
  mutex_lock(&inet_diag_mutex);
  netlink_rcv_skb(skb, &inet_diag_rcv_msg);
    if (inet_diag_table[nlh->nlmsg_type] ==
                        NULL) /* false handler is still registered */
    ...
    netlink_dump_start(idiagnl, skb, nlh,
                       inet_diag_dump, NULL);
          cb = kzalloc(sizeof(*cb), GFP_KERNEL);
               /* sleep here freeing memory
                * or preempt
                * or sleep later on nlk->cb_mutex
                */
                                          spin_lock(&inet_diag_register_lock);
                                          inet_diag_table[type] = NULL;
    ...                                   spin_unlock(&inet_diag_register_lock);
                                          synchronize_rcu();
                                          /* CPU1 is sleeping - RCU quiescent
                                           * state is passed
                                           */
                                          return;
    /* inet_diag_dump is finally called: */
    inet_diag_dump()
      handler = inet_diag_table[cb->nlh->nlmsg_type];
      BUG_ON(handler == NULL);
      /* OOPS! While we slept the unregister has set
       * handler to NULL :(
       */
```

Grep showed, that the register/unregister functions are called
from init/fini module callbacks for tcp_/dccp_diag, so it's OK
to use the inet_diag_mutex to synchronize manipulations with the
inet_diag_table and the access to it.

Besides, as Herbert pointed out, asynchronous dumps should hold
this mutex as well, and thus, we provide the mutex as cb_mutex one.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/ipv4/inet_diag.c b/net/ipv4/inet_diag.c
index b017073..6b3fffb 100644
--- a/net/ipv4/inet_diag.c
+++ b/net/ipv4/inet_diag.c
@@ -853,8 +853,6 @@ static void inet_diag_rcv(struct sk_buff *skb)
  mutex_unlock(&inet_diag_mutex);
 }

-static DEFINE_SPINLOCK(inet_diag_register_lock);
-
 int inet_diag_register(const struct inet_diag_handler *h)
 {
  const __u16 type = h->idiag_type;
@@ -863,13 +861,13 @@ int inet_diag_register(const struct inet_diag_handler *h)
  if (type >= INET_DIAG_GETSOCK_MAX)
   goto out;

- spin_lock(&inet_diag_register_lock);
+ mutex_lock(&inet_diag_mutex);
  err = -EEXIST;
  if (inet_diag_table[type] == NULL) {
   inet_diag_table[type] = h;
   err = 0;
  }
- spin_unlock(&inet_diag_register_lock);
+ mutex_unlock(&inet_diag_mutex);
 out:
  return err;
 }
@@ -882,11 +880,9 @@ void inet_diag_unregister(const struct inet_diag_handler *h)
  if (type >= INET_DIAG_GETSOCK_MAX)
   return;

- spin_lock(&inet_diag_register_lock);
+ mutex_lock(&inet_diag_mutex);
  inet_diag_table[type] = NULL;
- spin_unlock(&inet_diag_register_lock);
-
- synchronize_rcu();
+ mutex_unlock(&inet_diag_mutex);
 }
 EXPORT_SYMBOL_GPL(inet_diag_unregister);

@@ -901,7 +897,7 @@ static int __init inet_diag_init(void)
   goto out;

  idiagnl = netlink_kernel_create(&init_net, NETLINK_INET_DIAG, 0,
```

```
-     inet_diag_rcv, NULL, THIS_MODULE);
+   inet_diag_rcv, &inet_diag_mutex, THIS_MODULE);
  if (idiagnl == NULL)
   goto out_free_table;
  err = 0;
```

Subject: Re: [PATCH (resubmit)] Fix inet_diag.ko register vs rcv race
Posted by Herbert Xu on Thu, 29 Nov 2007 13:10:15 GMT
View Forum Message <> Reply to Message

On Thu, Nov 29, 2007 at 04:01:25PM +0300, Pavel Emelyanov wrote:
>
> Besides, as Herbert pointed out, asynchronous dumps should hold
> this mutex as well, and thus, we provide the mutex as cb_mutex one.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Thanks for the quick response! Patch applied to net-2.6.
--
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

Subject: Re: [PATCH (resubmit)] Fix inet_diag.ko register vs rcv race
Posted by Herbert Xu on Mon, 03 Dec 2007 04:56:22 GMT
View Forum Message <> Reply to Message

On Thu, Nov 29, 2007 at 04:01:25PM +0300, Pavel Emelyanov wrote:
>
> @@ -863,13 +861,13 @@ int inet_diag_register(const struct inet_diag_handler *h)
>   if (type >= INET_DIAG_GETSOCK_MAX)
>   goto out;
>
> - spin_lock(&inet_diag_register_lock);
> + mutex_lock(&inet_diag_mutex);
>   err = -EEXIST;
>   if (inet_diag_table[type] == NULL) {
>   inet_diag_table[type] = h;
>   err = 0;
>   }
> - spin_unlock(&inet_diag_register_lock);
> + mutex_unlock(&inet_diag_mutex);

Actually this causes a dead-lock when the handlers are built as modules

because we try to load them with that mutex held.

I've fixed it with this patch on top.

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
--
commit d523a328fb0271e1a763e985a21f2488fd816e7e
Author: Herbert Xu <herbert@gondor.apana.org.au>
Date: Mon Dec 3 15:51:25 2007 +1100

    [INET]: Fix inet_diag dead-lock regression

    The inet_diag register fix broke inet_diag module loading because the
    loaded module had to take the same mutex that's already held by the
    loader in order to register the new handler.

    This patch fixes it by introducing a separate mutex to protect the
    handling of handlers.

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

```
diff --git a/net/ipv4/inet_diag.c b/net/ipv4/inet_diag.c
index 6b3fffb..e468e7a 100644
--- a/net/ipv4/inet_diag.c
+++ b/net/ipv4/inet_diag.c
@@ -51,6 +51,29 @@ static struct sock *idiagnl;
 #define INET_DIAG_PUT(skb, attrtype, attrlen) \
  RTA_DATA(__RTA_PUT(skb, attrtype, attrlen))

+static DEFINE_MUTEX(inet_diag_table_mutex);
+
+static const struct inet_diag_handler *inet_diag_lock_handler(int type)
+{
+#ifdef CONFIG_KMOD
+ if (!inet_diag_table[type])
+  request_module("net-pf-%d-proto-%d-type-%d", PF_NETLINK,
+          NETLINK_INET_DIAG, type);
+#endif
+
+ mutex_lock(&inet_diag_table_mutex);
+ if (!inet_diag_table[type])
+  return ERR_PTR(-ENOENT);
+
```

```
+ return inet_diag_table[type];
+}
+
+static inline void inet_diag_unlock_handler(
+ const struct inet_diag_handler *handler)
+{
+ mutex_unlock(&inet_diag_table_mutex);
+}
+
 static int inet_csk_diag_fill(struct sock *sk,
         struct sk_buff *skb,
         int ext, u32 pid, u32 seq, u16 nlmsg_flags,
@@ -235,9 +258,12 @@ static int inet_diag_get_exact(struct sk_buff *in_skb,
  struct inet_hashinfo *hashinfo;
  const struct inet_diag_handler *handler;

- handler = inet_diag_table[nlh->nlmsg_type];
- BUG_ON(handler == NULL);
+ handler = inet_diag_lock_handler(nlh->nlmsg_type);
+ if (!handler)
+  return -ENOENT;
+
  hashinfo = handler->idiag_hashinfo;
+ err = -EINVAL;

  if (req->idiag_family == AF_INET) {
   sk = inet_lookup(hashinfo, req->id.idiag_dst[0],
@@ -255,11 +281,12 @@ static int inet_diag_get_exact(struct sk_buff *in_skb,
  }
 #endif
  else {
-  return -EINVAL;
+  goto unlock;
  }

+ err = -ENOENT;
  if (sk == NULL)
-  return -ENOENT;
+  goto unlock;

  err = -ESTALE;
  if ((req->id.idiag_cookie[0] != INET_DIAG_NOCOOKIE ||
@@ -296,6 +323,8 @@ out:
   else
    sock_put(sk);
  }
+unlock:
+ inet_diag_unlock_handler(handler);
```

```
  return err;
 }

@@ -678,8 +707,10 @@ static int inet_diag_dump(struct sk_buff *skb, struct netlink_callback
*cb)
  const struct inet_diag_handler *handler;
  struct inet_hashinfo *hashinfo;

- handler = inet_diag_table[cb->nlh->nlmsg_type];
- BUG_ON(handler == NULL);
+ handler = inet_diag_lock_handler(cb->nlh->nlmsg_type);
+ if (!handler)
+  goto no_handler;
+
  hashinfo = handler->idiag_hashinfo;

  s_i = cb->args[1];
@@ -743,7 +774,7 @@ skip_listen_ht:
  }

  if (!(r->idiag_states & ~(TCPF_LISTEN | TCPF_SYN_RECV)))
-  return skb->len;
+  goto unlock;

  for (i = s_i; i < hashinfo->ehash_size; i++) {
   struct inet_ehash_bucket *head = &hashinfo->ehash[i];
@@ -805,6 +836,9 @@ next_dying:
 done:
  cb->args[1] = i;
  cb->args[2] = num;
+unlock:
+ inet_diag_unlock_handler(handler);
+no_handler:
  return skb->len;
 }

@@ -816,15 +850,6 @@ static int inet_diag_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
      nlmsg_len(nlh) < hdrlen)
   return -EINVAL;

-#ifdef CONFIG_KMOD
- if (inet_diag_table[nlh->nlmsg_type] == NULL)
-  request_module("net-pf-%d-proto-%d-type-%d", PF_NETLINK,
-          NETLINK_INET_DIAG, nlh->nlmsg_type);
-#endif
-
- if (inet_diag_table[nlh->nlmsg_type] == NULL)
-  return -ENOENT;
```

```
-
  if (nlh->nlmsg_flags & NLM_F_DUMP) {
   if (nlmsg_attrlen(nlh, hdrlen)) {
    struct nlattr *attr;
@@ -861,13 +886,13 @@ int inet_diag_register(const struct inet_diag_handler *h)
  if (type >= INET_DIAG_GETSOCK_MAX)
   goto out;

- mutex_lock(&inet_diag_mutex);
+ mutex_lock(&inet_diag_table_mutex);
  err = -EEXIST;
  if (inet_diag_table[type] == NULL) {
   inet_diag_table[type] = h;
   err = 0;
  }
- mutex_unlock(&inet_diag_mutex);
+ mutex_unlock(&inet_diag_table_mutex);
 out:
  return err;
 }
@@ -880,9 +905,9 @@ void inet_diag_unregister(const struct inet_diag_handler *h)
  if (type >= INET_DIAG_GETSOCK_MAX)
   return;

- mutex_lock(&inet_diag_mutex);
+ mutex_lock(&inet_diag_table_mutex);
  inet_diag_table[type] = NULL;
- mutex_unlock(&inet_diag_mutex);
+ mutex_unlock(&inet_diag_table_mutex);
 }
 EXPORT_SYMBOL_GPL(inet_diag_unregister);

@@ -897,7 +922,7 @@ static int __init inet_diag_init(void)
   goto out;

  idiagnl = netlink_kernel_create(&init_net, NETLINK_INET_DIAG, 0,
-   inet_diag_rcv, &inet_diag_mutex, THIS_MODULE);
+     inet_diag_rcv, NULL, THIS_MODULE);
  if (idiagnl == NULL)
   goto out_free_table;
  err = 0;
```

Subject: Re: [PATCH (resubmit)] Fix inet_diag.ko register vs rcv race
Posted by Pavel Emelianov on Mon, 03 Dec 2007 09:01:06 GMT
View Forum Message <> Reply to Message

Herbert Xu wrote:

> On Thu, Nov 29, 2007 at 04:01:25PM +0300, Pavel Emelyanov wrote:
>> @@ -863,13 +861,13 @@ int inet_diag_register(const struct inet_diag_handler *h)
>>   if (type >= INET_DIAG_GETSOCK_MAX)
>>   goto out;
>>
>> - spin_lock(&inet_diag_register_lock);
>> + mutex_lock(&inet_diag_mutex);
>>   err = -EEXIST;
>>   if (inet_diag_table[type] == NULL) {
>>   inet_diag_table[type] = h;
>>   err = 0;
>>   }
>> - spin_unlock(&inet_diag_register_lock);
>> + mutex_unlock(&inet_diag_mutex);
>
> Actually this causes a dead-lock when the handlers are built as modules
> because we try to load them with that mutex held.

Ouch! Sorry, I didn't notice this :(

> I've fixed it with this patch on top.

Thanks!

> Cheers,

Pavel
