Subject: Re: [PATCH] Teach cifs about network namespaces.
Posted by Rob Landley on Tue, 11 Jan 2011 14:05:38 GMT
View Forum Message <> Reply to Message

On 01/11/2011 01:12 AM, Matt Helsley wrote:
> On Mon, Jan 10, 2011 at 10:35:19PM -0600, Rob Landley wrote:
>> From: Rob Landley <rlandley@parallels.com>
>>
>> Teach cifs about network namespaces, so mounting uses adresses and
>> routing visible from a container rather than from init context.
>>
>> For a long drawn out test reproduction sequence, see:
>>
>> http://landley.livejournal.com/47024.html
>> http://landley.livejournal.com/47205.html
>> http://landley.livejournal.com/47476.html
>>
>> Signed-off-by: Rob Landley <rlandley@parallels.com>
>> ---
>>
>> fs/cifs/cifsglob.h | 32 ++++++++++++++++++++++++++++++++
>> fs/cifs/connect.c | 22 ++++++++++++++++-----
>> 2 files changed, 49 insertions(+), 5 deletions(-)
>>
>> diff --git a/fs/cifs/cifsglob.h b/fs/cifs/cifsglob.h
>> index 7136c0c..86f31bb 100644
>> --- a/fs/cifs/cifsglob.h
>> +++ b/fs/cifs/cifsglob.h
>> @@ -168,6 +168,9 @@ struct TCP_Server_Info {
>>   struct sockaddr_in6 sockAddr6;
>> } addr;
>> struct sockaddr_storage srcaddr; /* locally bind to this IP */
>> +#ifdef CONFIG_NET_NS
>> + struct net *net;
>> +#endif
>
> I'm assuming this bit is correct -- don't know enough about CIFS to be
> sure...
>
>> wait_queue_head_t response_q;
>> wait_queue_head_t request_q; /* if more than maxmpx to srvr must block*/
>> struct list_head pending_mid_q;
>> @@ -227,6 +230,35 @@ struct TCP_Server_Info {
>> };
>>
>> /*
>> + * Macros to allow the TCP_Server_Info->net field and related code to drop out
>> + * when CONFIG_NET_NS isn't set.

>> + */
>> +
>> +static inline struct net *
>> +cifs_net_ns(struct TCP_Server_Info *srv)
>> +{
>> +#ifdef CONFIG_NET_NS
>> + return srv->net;
>> +#else
>> + return &init_net;
>> +#endif
>> +}
>
> I thought style dictated we do this a different way:
>
> #ifdef CONFIG_NET_NS
> static inline struct net * cifs_net_ns(struct TCP_Server_Info *srv)
> {
> return srv->net;
> }
> <other CONFIG_NET_NS cases>
> #else
> static inline struct net * cifs_net_ns(struct TCP_Server_Info *srv)
> {
> return &init_net;
> }
> <other no-ops>
> #endif /* CONFIG_NET_NS */

If you want to duplicate more code and open the possibility of the
declarations mismatching if the config changes. *shrug*

>> +
>> +static inline void
>> +cifs_set_net_ns(struct TCP_Server_Info *srv, struct net *net)
>> +{
>> +#ifdef CONFIG_NET_NS
>> + srv->net = net;
>> +#endif
>> +}
>> +
>> +#ifdef CONFIG_NET_NS
>> +#define cifs_use_net_ns() (1)
>> +#else
>> +#define cifs_use_net_ns() (0)
>> +#endif
>
> This looks wrong -- we shouldn't need this at all. The #ifdef bits in
> your patch already make all the cases below become empty/no-ops when

> CONFIG_NET_NS=n.

Except that bloat-o-meter said they were still generating code and
making the kernel bigger. (Things like calling get() and put() on
init_net to twiddle its reference count.)

I'm a long-time embedded programmer, I try not to make the code bigger
than necessary, especially when we have a config symbol to remove stuff.
 I did ponder turning those into HAVE_NET_HS so the if was more
obviously against a constant.

>> +
>> +/*
>>   * Session structure. One of these for each uid session with a particular host
>>   */
>> struct cifsSesInfo {
>> diff --git a/fs/cifs/connect.c b/fs/cifs/connect.c
>> index cc1a860..b4faef0 100644
>> --- a/fs/cifs/connect.c
>> +++ b/fs/cifs/connect.c
>> @@ -1545,6 +1545,10 @@ cifs_find_tcp_session(struct sockaddr *addr, struct smb_vol *vol)
>>
>>   spin_lock(&cifs_tcp_ses_lock);
>>   list_for_each_entry(server, &cifs_tcp_ses_list, tcp_ses_list) {
>> + if (cifs_use_net_ns()
>> +   && cifs_net_ns(server) == current->nsproxy->net_ns)
>> +   continue;
>
> This looks wrong -- you want to invert part of this I think (and drop the
> unnecessary cifs_use_net_ns()):

You're right, I got the test backwards. Thanks.

The reason for the guards is that compiler couldn't tell that
current->nsproxy->net_ns always contains the same value, so without a
test against a constant allowing it to do dead code elimination, it will
generate code to perform the useless test.

> if (cifs_net_ns(server) != current->nsproxy->net_ns)
>   continue;
>
> This is obvious when you note that the context below shows that we
> 'continue' to the next entry when the addresses don't match:
>
>> +
>>   if (!match_address(server, addr,
>>   (struct sockaddr *)&vol->srcaddr))
>>   continue;

>> @@ -1572,6 +1576,9 @@ cifs_put_tcp_session(struct TCP_Server_Info *server)
>>   return;
>>   }
>>
>> + if (cifs_use_net_ns())
>> + put_net(cifs_net_ns(server));
>> +
>
> I think this should just be:
>
> put_net(cifs_net_ns(server));

Hmmm... that one you're probably right, because
include/net/net_namespace.h makes put_net() an empty inline, so as long
as cifs_net_ns() has no side effects the compiler should be able to drop
the whole thing out. (Whether or not it will, I have to test...)

>>   list_del_init(&server->tcp_ses_list);
>>   spin_unlock(&cifs_tcp_ses_lock);
>>
>> @@ -1677,6 +1684,9 @@ cifs_get_tcp_session(struct smb_vol *volume_info)
>>   sizeof(tcp_ses->srcaddr));
>>   ++tcp_ses->srv_count;
>>
>> + if (cifs_use_net_ns())
>> + cifs_set_net_ns(tcp_ses, get_net(current->nsproxy->net_ns));
>> +
>
> Just use cifs_set_net_ns() because it already turns into a no-op when
> CONFIG_NET_NS=n

no-op yes. no-code, I want to make sure. (I tried it with just the
macros the first time through, and scripts/bloat-o-meter kept saying
code was being generated. I'll fire up objdump and see if the
disassembly tells me anything...)

>>   if (addr.ss_family == AF_INET6) {
>>   cFYI(1, "attempting ipv6 connect");
>>   /* BB should we allow ipv6 on port 139? */
>> @@ -1720,6 +1730,9 @@ cifs_get_tcp_session(struct smb_vol *volume_info)
>> out_err_crypto_release:
>>   cifs_crypto_shash_release(tcp_ses);
>>
>> + if (cifs_use_net_ns())
>> + put_net(cifs_net_ns(tcp_ses));
>> +
>
> etc.
>

```
>> +
>>  out_err:
>>   if (tcp_ses) {
>>    if (!IS_ERR(tcp_ses->hostname))
>> @@ -2145,8 +2158,8 @@ ipv4_connect(struct TCP_Server_Info *server)
>>   struct socket *socket = server->ssocket;
>>
>>   if (socket == NULL) {
>> -  rc = sock_create_kern(PF_INET, SOCK_STREAM,
>> -          IPPROTO_TCP, &socket);
>> +  rc = __sock_create(cifs_net_ns(server), PF_INET,
>> +        SOCK_STREAM, IPPROTO_TCP, &socket, 1);
>>    if (rc < 0) {
>>     cERROR(1, "Error %d creating socket", rc);
>>     return rc;
>> @@ -2310,11 +2323,10 @@ ipv6_connect(struct TCP_Server_Info *server)
>>   struct socket *socket = server->ssocket;
>>
>>   if (socket == NULL) {
>> -  rc = sock_create_kern(PF_INET6, SOCK_STREAM,
>> -          IPPROTO_TCP, &socket);
>> +  rc = __sock_create(cifs_net_ns(server), PF_INET6,
>> +        SOCK_STREAM, IPPROTO_TCP, &socket, 1);
>>    if (rc < 0) {
>>     cERROR(1, "Error %d creating ipv6 socket", rc);
>> -   socket = NULL;
>>     return rc;
>>    }
```

Note, those two add 16 bytes each on x86-64 (two extra 8-byte arguments), but that I couldn't easily stop. I might try to merge the ipv4/ipv6 functions, but that's a separate patch.

Rob

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs