Subject: [PATCH 1/3] netns netfilter: semi-rewrite of /proc/net/foo_tables_*
Posted by Alexey Dobriyan on Fri, 25 Jan 2008 16:43:14 GMT
View Forum Message <> Reply to Message

Argh, there are many small but still wrong things with /proc/net/*_tables_*
so I decided to do overhaul simultaneously making it more suitable for
per-netns /proc/net/*_tables_* implementation.

Fix
a) xt_get_idx() duplicating now standard seq_list_start/seq_list_next
   iterators
b) tables/matches/targets list was chosen again and again on every ->next
c) multiple useless "af >= NPROTO" checks -- we simple don't supply invalid
   AFs there and registration function should BUG_ON instead.

   Regardless, the one in ->next() is the most useless -- ->next doesn't
   run at all if ->start fails.
d) Don't use mutex_lock_interruptible() -- it can fail and ->stop is
   executed even if ->start failed, so unlock without lock is possible.

As side effect, streamline code by splitting xt_tgt_ops into xt_target_ops,
xt_matches_ops, xt_tables_ops.

xt_tables_ops hooks will be changed by per-netns code. Code of
xt_matches_ops, xt_target_ops is identical except the list chosen for
iterating, but I think consolidating code for two files not worth it
given "<< 16" hacks needed for it.

Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>
---

 net/netfilter/x_tables.c | 224 ++++++++++++++++++++++++++++++++-----------------
 1 file changed, 145 insertions(+), 79 deletions(-)

--- a/net/netfilter/x_tables.c
+++ b/net/netfilter/x_tables.c
@@ -726,124 +726,190 @@ void *xt_unregister_table(struct xt_table *table)
 EXPORT_SYMBOL_GPL(xt_unregister_table);

 #ifdef CONFIG_PROC_FS
-static struct list_head *xt_get_idx(struct list_head *list, struct seq_file *seq, loff_t pos)
+static void *xt_table_seq_start(struct seq_file *seq, loff_t *pos)
 {
- struct list_head *head = list->next;
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
+ u_int16_t af = (unsigned long)pde->data;

- if (!head || list_empty(list))

```
-  return NULL;
+ mutex_lock(&xt[af].mutex);
+ return seq_list_start(&init_net.xt.tables[af], *pos);
+}

- while (pos && (head = head->next)) {
-  if (head == list)
-   return NULL;
-  pos--;
- }
- return pos ? NULL : head;
-}
-
-static struct list_head *type2list(u_int16_t af, u_int16_t type)
-{
- struct list_head *list;
-
- switch (type) {
- case TARGET:
-  list = &xt[af].target;
-  break;
- case MATCH:
-  list = &xt[af].match;
-  break;
- case TABLE:
-  list = &init_net.xt.tables[af];
-  break;
- default:
-  list = NULL;
-  break;
- }
+static void *xt_table_seq_next(struct seq_file *seq, void *v, loff_t *pos)
+{
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
+ u_int16_t af = (unsigned long)pde->data;

- return list;
+ return seq_list_next(v, &init_net.xt.tables[af], pos);
 }

-static void *xt_tgt_seq_start(struct seq_file *seq, loff_t *pos)
+static void xt_table_seq_stop(struct seq_file *seq, void *v)
 {
- struct proc_dir_entry *pde = (struct proc_dir_entry *) seq->private;
- u_int16_t af = (unsigned long)pde->data & 0xffff;
- u_int16_t type = (unsigned long)pde->data >> 16;
- struct list_head *list;
+ struct proc_dir_entry *pde = seq->private;
```

```
+ u_int16_t af = (unsigned long)pde->data;

- if (af >= NPROTO)
-  return NULL;
+ mutex_unlock(&xt[af].mutex);
+}

- list = type2list(af, type);
- if (!list)
-  return NULL;
+static int xt_table_seq_show(struct seq_file *seq, void *v)
+{
+ struct xt_table *table = list_entry(v, struct xt_table, list);

- if (mutex_lock_interruptible(&xt[af].mutex) != 0)
-  return NULL;
+ if (strlen(table->name))
+  return seq_printf(seq, "%s\n", table->name);
+ else
+  return 0;
+}
+
+static const struct seq_operations xt_table_seq_ops = {
+ .start = xt_table_seq_start,
+ .next = xt_table_seq_next,
+ .stop = xt_table_seq_stop,
+ .show = xt_table_seq_show,
+};
+
+static int xt_table_open(struct inode *inode, struct file *file)
+{
+ int ret;
+
+ ret = seq_open(file, &xt_table_seq_ops);
+ if (!ret) {
+  struct seq_file *seq = file->private_data;

- return xt_get_idx(list, seq, *pos);
+  seq->private = PDE(inode);
+ }
+ return ret;
 }

-static void *xt_tgt_seq_next(struct seq_file *seq, void *v, loff_t *pos)
+static const struct file_operations xt_table_ops = {
+ .owner  = THIS_MODULE,
+ .open  = xt_table_open,
+ .read  = seq_read,
```

```
+ .llseek  = seq_lseek,
+ .release = seq_release,
+};
+
+static void *xt_match_seq_start(struct seq_file *seq, loff_t *pos)
 {
- struct proc_dir_entry *pde = seq->private;
- u_int16_t af = (unsigned long)pde->data & 0xffff;
- u_int16_t type = (unsigned long)pde->data >> 16;
- struct list_head *list;
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
+ u_int16_t af = (unsigned long)pde->data;

- if (af >= NPROTO)
-  return NULL;
+ mutex_lock(&xt[af].mutex);
+ return seq_list_start(&xt[af].match, *pos);
+}

- list = type2list(af, type);
- if (!list)
-  return NULL;
+static void *xt_match_seq_next(struct seq_file *seq, void *v, loff_t *pos)
+{
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
+ u_int16_t af = (unsigned long)pde->data;

- (*pos)++;
- return xt_get_idx(list, seq, *pos);
+ return seq_list_next(v, &xt[af].match, pos);
 }

-static void xt_tgt_seq_stop(struct seq_file *seq, void *v)
+static void xt_match_seq_stop(struct seq_file *seq, void *v)
 {
  struct proc_dir_entry *pde = seq->private;
- u_int16_t af = (unsigned long)pde->data & 0xffff;
+ u_int16_t af = (unsigned long)pde->data;

  mutex_unlock(&xt[af].mutex);
 }

-static int xt_name_seq_show(struct seq_file *seq, void *v)
+static int xt_match_seq_show(struct seq_file *seq, void *v)
 {
- char *name = (char *)v + sizeof(struct list_head);
+ struct xt_match *match = list_entry(v, struct xt_match, list);
```

```
- if (strlen(name))
-  return seq_printf(seq, "%s\n", name);
+ if (strlen(match->name))
+  return seq_printf(seq, "%s\n", match->name);
  else
   return 0;
 }

-static const struct seq_operations xt_tgt_seq_ops = {
- .start = xt_tgt_seq_start,
- .next = xt_tgt_seq_next,
- .stop = xt_tgt_seq_stop,
- .show = xt_name_seq_show,
+static const struct seq_operations xt_match_seq_ops = {
+ .start = xt_match_seq_start,
+ .next = xt_match_seq_next,
+ .stop = xt_match_seq_stop,
+ .show = xt_match_seq_show,
 };

-static int xt_tgt_open(struct inode *inode, struct file *file)
+static int xt_match_open(struct inode *inode, struct file *file)
 {
  int ret;

- ret = seq_open(file, &xt_tgt_seq_ops);
+ ret = seq_open(file, &xt_match_seq_ops);
  if (!ret) {
   struct seq_file *seq = file->private_data;
-  struct proc_dir_entry *pde = PDE(inode);

-  seq->private = pde;
+  seq->private = PDE(inode);
  }
+ return ret;
+}
+
+static const struct file_operations xt_match_ops = {
+ .owner  = THIS_MODULE,
+ .open  = xt_match_open,
+ .read  = seq_read,
+ .llseek  = seq_lseek,
+ .release = seq_release,
+};
+
+static void *xt_target_seq_start(struct seq_file *seq, loff_t *pos)
+{
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
```

```
+ u_int16_t af = (unsigned long)pde->data;
+
+ mutex_lock(&xt[af].mutex);
+ return seq_list_start(&xt[af].target, *pos);
+}
+
+static void *xt_target_seq_next(struct seq_file *seq, void *v, loff_t *pos)
+{
+ struct proc_dir_entry *pde = (struct proc_dir_entry *)seq->private;
+ u_int16_t af = (unsigned long)pde->data;
+
+ return seq_list_next(v, &xt[af].target, pos);
+}
+
+static void xt_target_seq_stop(struct seq_file *seq, void *v)
+{
+ struct proc_dir_entry *pde = seq->private;
+ u_int16_t af = (unsigned long)pde->data;
+
+ mutex_unlock(&xt[af].mutex);
+}
+
+static int xt_target_seq_show(struct seq_file *seq, void *v)
+{
+ struct xt_target *target = list_entry(v, struct xt_target, list);
+
+ if (strlen(target->name))
+  return seq_printf(seq, "%s\n", target->name);
+ else
+  return 0;
+}
+
+static const struct seq_operations xt_target_seq_ops = {
+ .start = xt_target_seq_start,
+ .next = xt_target_seq_next,
+ .stop = xt_target_seq_stop,
+ .show = xt_target_seq_show,
+};
+
+static int xt_target_open(struct inode *inode, struct file *file)
+{
+ int ret;

+ ret = seq_open(file, &xt_target_seq_ops);
+ if (!ret) {
+  struct seq_file *seq = file->private_data;
+
+  seq->private = PDE(inode);
```

```
+ }
  return ret;
 }

-static const struct file_operations xt_file_ops = {
+static const struct file_operations xt_target_ops = {
  .owner  = THIS_MODULE,
- .open  = xt_tgt_open,
+ .open  = xt_target_open,
  .read  = seq_read,
  .llseek  = seq_lseek,
  .release = seq_release,
@@ -869,25 +935,25 @@ int xt_proto_init(int af)
 #ifdef CONFIG_PROC_FS
  strlcpy(buf, xt_prefix[af], sizeof(buf));
  strlcat(buf, FORMAT_TABLES, sizeof(buf));
- proc = proc_net_fops_create(&init_net, buf, 0440, &xt_file_ops);
+ proc = proc_net_fops_create(&init_net, buf, 0440, &xt_table_ops);
  if (!proc)
   goto out;
- proc->data = (void *) ((unsigned long) af | (TABLE << 16));
+ proc->data = (void *)(unsigned long)af;


  strlcpy(buf, xt_prefix[af], sizeof(buf));
  strlcat(buf, FORMAT_MATCHES, sizeof(buf));
- proc = proc_net_fops_create(&init_net, buf, 0440, &xt_file_ops);
+ proc = proc_net_fops_create(&init_net, buf, 0440, &xt_match_ops);
  if (!proc)
   goto out_remove_tables;
- proc->data = (void *) ((unsigned long) af | (MATCH << 16));
+ proc->data = (void *)(unsigned long)af;

  strlcpy(buf, xt_prefix[af], sizeof(buf));
  strlcat(buf, FORMAT_TARGETS, sizeof(buf));
- proc = proc_net_fops_create(&init_net, buf, 0440, &xt_file_ops);
+ proc = proc_net_fops_create(&init_net, buf, 0440, &xt_target_ops);
  if (!proc)
   goto out_remove_matches;
- proc->data = (void *) ((unsigned long) af | (TARGET << 16));
+ proc->data = (void *)(unsigned long)af;
 #endif

  return 0;
```

Subject: Re: [PATCH 1/3] netns netfilter: semi-rewrite of /proc/net/foo_tables_*

Posted by Patrick McHardy on Wed, 30 Jan 2008 15:17:10 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> Argh, there are many small but still wrong things with /proc/net/*_tables_*
> so I decided to do overhaul simultaneously making it more suitable for
> per-netns /proc/net/*_tables_* implementation.
>
> Fix
> a) xt_get_idx() duplicating now standard seq_list_start/seq_list_next
>   iterators
> b) tables/matches/targets list was chosen again and again on every ->next
> c) multiple useless "af >= NPROTO" checks -- we simple don't supply invalid
>   AFs there and registration function should BUG_ON instead.
>
>   Regardless, the one in ->next() is the most useless -- ->next doesn't
>   run at all if ->start fails.
> d) Don't use mutex_lock_interruptible() -- it can fail and ->stop is
>   executed even if ->start failed, so unlock without lock is possible.
>
> As side effect, streamline code by splitting xt_tgt_ops into xt_target_ops,
> xt_matches_ops, xt_tables_ops.
>
> xt_tables_ops hooks will be changed by per-netns code. Code of
> xt_matches_ops, xt_target_ops is identical except the list chosen for
> iterating, but I think consolidating code for two files not worth it
> given "<< 16" hacks needed for it.
>
> Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>

Applied, and I also removed the now unused TABLE/TARGET/MATCH enum.