Subject: [PATCH 3/5] netns netfilter: return new table from {arp, ip, ip6}t_register_table()
Posted by Alexey Dobriyan on Mon, 21 Jan 2008 14:53:57 GMT
View Forum Message <> Reply to Message

Typical table module registers xt_table structure (i.e. packet_filter)
and link it to list during it. We can't use one template for it because
corresponding list_head will become corrupted. We also can't unregister
with template because it wasn't changed at all and thus doesn't know in
which list it is.

So, we duplicate template at the very first step of table registration.
Table modules will save it for use during unregistration time and actual
filtering.

Do it at once to not screw bisection.

P.S.: renaming i.e. packet_filter => __packet_filter is temporary until
full netnsization of table modules is done.

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

include/linux/netfilter_arp/arp_tables.h | 4 ++--
include/linux/netfilter_ipv4/ip_tables.h | 5 +++--
include/linux/netfilter_ipv6/ip6_tables.h | 4 ++--
net/ipv4/netfilter/arp_tables.c | 22 ++++++++++++----------
net/ipv4/netfilter/arptable_filter.c | 15 ++++++++-------
net/ipv4/netfilter/ip_tables.c | 28 ++++++++++++++++++----------
net/ipv4/netfilter/iptable_filter.c | 18 ++++++++++--------
net/ipv4/netfilter/iptable_mangle.c | 18 ++++++++++--------
net/ipv4/netfilter/iptable_raw.c | 18 ++++++++++--------
net/ipv4/netfilter/nf_nat_rule.c | 16 ++++++++--------
net/ipv6/netfilter/ip6_tables.c | 24 ++++++++++++++----------
net/ipv6/netfilter/ip6table_filter.c | 17 +++++++++--------
net/ipv6/netfilter/ip6table_mangle.c | 17 +++++++++--------
net/ipv6/netfilter/ip6table_raw.c | 15 ++++++++-------
net/netfilter/x_tables.c | 13 +++++++++--
15 files changed, 134 insertions(+), 100 deletions(-)

--- a/include/linux/netfilter_arp/arp_tables.h
+++ b/include/linux/netfilter_arp/arp_tables.h
@@ -271,8 +271,8 @@ struct arpt_error
xt_register_target(tgt); })
#define arpt_unregister_target(tgt) xt_unregister_target(tgt)

-extern int arpt_register_table(struct arpt_table *table,
-  const struct arpt_replace *repl);

```
+extern struct arpt_table *arpt_register_table(struct arpt_table *table,
+          const struct arpt_replace *repl);
 extern void arpt_unregister_table(struct arpt_table *table);
 extern unsigned int arpt_do_table(struct sk_buff *skb,
       unsigned int hook,
--- a/include/linux/netfilter_ipv4/ip_tables.h
+++ b/include/linux/netfilter_ipv4/ip_tables.h
@@ -244,8 +244,9 @@ ipt_get_target(struct ipt_entry *e)
 #include <linux/init.h>
 extern void ipt_init(void) __init;

-extern int ipt_register_table(struct xt_table *table,
-         const struct ipt_replace *repl);
+extern struct xt_table *ipt_register_table(struct net *net,
+         struct xt_table *table,
+         const struct ipt_replace *repl);
 extern void ipt_unregister_table(struct xt_table *table);

 /* Standard entry. */
--- a/include/linux/netfilter_ipv6/ip6_tables.h
+++ b/include/linux/netfilter_ipv6/ip6_tables.h
@@ -305,8 +305,8 @@ ip6t_get_target(struct ip6t_entry *e)
 #include <linux/init.h>
 extern void ip6t_init(void) __init;

-extern int ip6t_register_table(struct xt_table *table,
-          const struct ip6t_replace *repl);
+extern struct xt_table *ip6t_register_table(struct xt_table *table,
+          const struct ip6t_replace *repl);
 extern void ip6t_unregister_table(struct xt_table *table);
 extern unsigned int ip6t_do_table(struct sk_buff *skb,
       unsigned int hook,
--- a/net/ipv4/netfilter/arp_tables.c
+++ b/net/ipv4/netfilter/arp_tables.c
@@ -1719,8 +1719,8 @@ static int do_arpt_get_ctl(struct sock *sk, int cmd, void __user *user,
int *len
  return ret;
 }

-int arpt_register_table(struct arpt_table *table,
-   const struct arpt_replace *repl)
+struct arpt_table *arpt_register_table(struct arpt_table *table,
+          const struct arpt_replace *repl)
 {
  int ret;
  struct xt_table_info *newinfo;
@@ -1732,7 +1732,7 @@ int arpt_register_table(struct arpt_table *table,
  newinfo = xt_alloc_table_info(repl->size);
```

```
  if (!newinfo) {
   ret = -ENOMEM;
-  return ret;
+  goto out;
  }

  /* choose the copy on our node/cpu */
@@ -1746,18 +1746,20 @@ int arpt_register_table(struct arpt_table *table,
          repl->underflow);

  duprintf("arpt_register_table: translate table gives %d\n", ret);
- if (ret != 0) {
-  xt_free_table_info(newinfo);
-  return ret;
- }
+ if (ret != 0)
+  goto out_free;

  new_table = xt_register_table(&init_net, table, &bootstrap, newinfo);
  if (IS_ERR(new_table)) {
-  xt_free_table_info(newinfo);
-  return PTR_ERR(new_table);
+  ret = PTR_ERR(new_table);
+  goto out_free;
  }
+ return new_table;

- return 0;
+out_free:
+ xt_free_table_info(newinfo);
+out:
+ return ERR_PTR(ret);
 }

 void arpt_unregister_table(struct arpt_table *table)
--- a/net/ipv4/netfilter/arptable_filter.c
+++ b/net/ipv4/netfilter/arptable_filter.c
@@ -45,7 +45,7 @@ static struct
  .term = ARPT_ERROR_INIT,
 };

-static struct arpt_table packet_filter = {
+static struct arpt_table __packet_filter = {
  .name  = "filter",
  .valid_hooks = FILTER_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
@@ -53,6 +53,7 @@ static struct arpt_table packet_filter = {
  .me  = THIS_MODULE,
```

```
  .af  = NF_ARP,
 };
+static struct arpt_table *packet_filter;

 /* The work comes in here from netfilter.c */
 static unsigned int arpt_hook(unsigned int hook,
@@ -61,7 +62,7 @@ static unsigned int arpt_hook(unsigned int hook,
         const struct net_device *out,
         int (*okfn)(struct sk_buff *))
 {
- return arpt_do_table(skb, hook, in, out, &packet_filter);
+ return arpt_do_table(skb, hook, in, out, packet_filter);
 }

 static struct nf_hook_ops arpt_ops[] __read_mostly = {
@@ -90,9 +91,9 @@ static int __init arptable_filter_init(void)
  int ret;

  /* Register table */
- ret = arpt_register_table(&packet_filter, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_filter = arpt_register_table(&__packet_filter, &initial_table.repl);
+ if (IS_ERR(packet_filter))
+  return PTR_ERR(packet_filter);

  ret = nf_register_hooks(arpt_ops, ARRAY_SIZE(arpt_ops));
  if (ret < 0)
@@ -100,14 +101,14 @@ static int __init arptable_filter_init(void)
  return ret;

 cleanup_table:
- arpt_unregister_table(&packet_filter);
+ arpt_unregister_table(packet_filter);
  return ret;
 }

 static void __exit arptable_filter_fini(void)
 {
  nf_unregister_hooks(arpt_ops, ARRAY_SIZE(arpt_ops));
- arpt_unregister_table(&packet_filter);
+ arpt_unregister_table(packet_filter);
 }

 module_init(arptable_filter_init);
--- a/net/ipv4/netfilter/ip_tables.c
+++ b/net/ipv4/netfilter/ip_tables.c
@@ -2048,7 +2048,8 @@ do_ipt_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
```

```
  return ret;
 }

-int ipt_register_table(struct xt_table *table, const struct ipt_replace *repl)
+struct xt_table *ipt_register_table(struct net *net, struct xt_table *table,
+        const struct ipt_replace *repl)
 {
  int ret;
  struct xt_table_info *newinfo;
@@ -2058,8 +2059,10 @@ int ipt_register_table(struct xt_table *table, const struct ipt_replace
*repl)
  struct xt_table *new_table;

  newinfo = xt_alloc_table_info(repl->size);
- if (!newinfo)
-  return -ENOMEM;
+ if (!newinfo) {
+  ret = -ENOMEM;
+  goto out;
+ }

  /* choose the copy on our node/cpu, but dont care about preemption */
  loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
@@ -2070,18 +2073,21 @@ int ipt_register_table(struct xt_table *table, const struct ipt_replace
*repl)
         repl->num_entries,
         repl->hook_entry,
         repl->underflow);
- if (ret != 0) {
-  xt_free_table_info(newinfo);
-  return ret;
- }
+ if (ret != 0)
+  goto out_free;

- new_table = xt_register_table(&init_net, table, &bootstrap, newinfo);
+ new_table = xt_register_table(net, table, &bootstrap, newinfo);
  if (IS_ERR(new_table)) {
-  xt_free_table_info(newinfo);
-  return PTR_ERR(new_table);
+  ret = PTR_ERR(new_table);
+  goto out_free;
  }

- return 0;
+ return new_table;
+
+out_free:
```

```
+ xt_free_table_info(newinfo);
+out:
+ return ERR_PTR(ret);
 }

 void ipt_unregister_table(struct xt_table *table)
--- a/net/ipv4/netfilter/iptable_filter.c
+++ b/net/ipv4/netfilter/iptable_filter.c
@@ -53,13 +53,14 @@ static struct
  .term = IPT_ERROR_INIT,   /* ERROR */
 };

-static struct xt_table packet_filter = {
+static struct xt_table __packet_filter = {
  .name  = "filter",
  .valid_hooks = FILTER_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
  .me  = THIS_MODULE,
  .af  = AF_INET,
 };
+static struct xt_table *packet_filter;

 /* The work comes in here from netfilter.c. */
 static unsigned int
@@ -69,7 +70,7 @@ ipt_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ipt_do_table(skb, hook, in, out, &packet_filter);
+ return ipt_do_table(skb, hook, in, out, packet_filter);
 }

 static unsigned int
@@ -88,7 +89,7 @@ ipt_local_out_hook(unsigned int hook,
   return NF_ACCEPT;
  }

- return ipt_do_table(skb, hook, in, out, &packet_filter);
+ return ipt_do_table(skb, hook, in, out, packet_filter);
 }

 static struct nf_hook_ops ipt_ops[] __read_mostly = {
@@ -132,9 +133,10 @@ static int __init iptable_filter_init(void)
  initial_table.entries[1].target.verdict = -forward - 1;

  /* Register table */
- ret = ipt_register_table(&packet_filter, &initial_table.repl);
- if (ret < 0)
```

```
-  return ret;
+ packet_filter = ipt_register_table(&init_net, &__packet_filter,
+        &initial_table.repl);
+ if (IS_ERR(packet_filter))
+  return PTR_ERR(packet_filter);

  /* Register hooks */
  ret = nf_register_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
@@ -144,14 +146,14 @@ static int __init iptable_filter_init(void)
  return ret;

  cleanup_table:
- ipt_unregister_table(&packet_filter);
+ ipt_unregister_table(packet_filter);
  return ret;
 }

 static void __exit iptable_filter_fini(void)
 {
  nf_unregister_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
- ipt_unregister_table(&packet_filter);
+ ipt_unregister_table(packet_filter);
 }

 module_init(iptable_filter_init);
--- a/net/ipv4/netfilter/iptable_mangle.c
+++ b/net/ipv4/netfilter/iptable_mangle.c
@@ -64,13 +64,14 @@ static struct
  .term = IPT_ERROR_INIT,   /* ERROR */
 };

-static struct xt_table packet_mangler = {
+static struct xt_table __packet_mangler = {
  .name  = "mangle",
  .valid_hooks = MANGLE_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
  .me  = THIS_MODULE,
  .af  = AF_INET,
 };
+static struct xt_table *packet_mangler;

 /* The work comes in here from netfilter.c. */
 static unsigned int
@@ -80,7 +81,7 @@ ipt_route_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ipt_do_table(skb, hook, in, out, &packet_mangler);
```

```
+ return ipt_do_table(skb, hook, in, out, packet_mangler);
 }

 static unsigned int
@@ -112,7 +113,7 @@ ipt_local_hook(unsigned int hook,
  daddr = iph->daddr;
  tos = iph->tos;

- ret = ipt_do_table(skb, hook, in, out, &packet_mangler);
+ ret = ipt_do_table(skb, hook, in, out, packet_mangler);
  /* Reroute for ANY change. */
  if (ret != NF_DROP && ret != NF_STOLEN && ret != NF_QUEUE) {
   iph = ip_hdr(skb);
@@ -171,9 +172,10 @@ static int __init iptable_mangle_init(void)
  int ret;

  /* Register table */
- ret = ipt_register_table(&packet_mangler, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_mangler = ipt_register_table(&init_net, &__packet_mangler,
+         &initial_table.repl);
+ if (IS_ERR(packet_mangler))
+  return PTR_ERR(packet_mangler);

  /* Register hooks */
  ret = nf_register_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
@@ -183,14 +185,14 @@ static int __init iptable_mangle_init(void)
  return ret;

  cleanup_table:
- ipt_unregister_table(&packet_mangler);
+ ipt_unregister_table(packet_mangler);
  return ret;
 }

 static void __exit iptable_mangle_fini(void)
 {
  nf_unregister_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
- ipt_unregister_table(&packet_mangler);
+ ipt_unregister_table(packet_mangler);
 }

 module_init(iptable_mangle_init);
--- a/net/ipv4/netfilter/iptable_raw.c
+++ b/net/ipv4/netfilter/iptable_raw.c
@@ -36,13 +36,14 @@ static struct
  .term = IPT_ERROR_INIT,   /* ERROR */
```

```
 };

-static struct xt_table packet_raw = {
+static struct xt_table __packet_raw = {
  .name = "raw",
  .valid_hooks =  RAW_VALID_HOOKS,
  .lock = RW_LOCK_UNLOCKED,
  .me = THIS_MODULE,
  .af = AF_INET,
 };
+static struct xt_table *packet_raw;

 /* The work comes in here from netfilter.c. */
 static unsigned int
@@ -52,7 +53,7 @@ ipt_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ipt_do_table(skb, hook, in, out, &packet_raw);
+ return ipt_do_table(skb, hook, in, out, packet_raw);
 }

 static unsigned int
@@ -70,7 +71,7 @@ ipt_local_hook(unsigned int hook,
           "packet.\n");
   return NF_ACCEPT;
  }
- return ipt_do_table(skb, hook, in, out, &packet_raw);
+ return ipt_do_table(skb, hook, in, out, packet_raw);
 }

 /* 'raw' is the very first table. */
@@ -96,9 +97,10 @@ static int __init iptable_raw_init(void)
  int ret;

  /* Register table */
- ret = ipt_register_table(&packet_raw, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_raw = ipt_register_table(&init_net, &__packet_raw,
+      &initial_table.repl);
+ if (IS_ERR(packet_raw))
+  return PTR_ERR(packet_raw);

  /* Register hooks */
  ret = nf_register_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
@@ -108,14 +110,14 @@ static int __init iptable_raw_init(void)
  return ret;
```

```
  cleanup_table:
- ipt_unregister_table(&packet_raw);
+ ipt_unregister_table(packet_raw);
  return ret;
 }

 static void __exit iptable_raw_fini(void)
 {
  nf_unregister_hooks(ipt_ops, ARRAY_SIZE(ipt_ops));
- ipt_unregister_table(&packet_raw);
+ ipt_unregister_table(packet_raw);
 }

 module_init(iptable_raw_init);
--- a/net/ipv4/netfilter/nf_nat_rule.c
+++ b/net/ipv4/netfilter/nf_nat_rule.c
@@ -58,13 +58,14 @@ static struct
  .term = IPT_ERROR_INIT,   /* ERROR */
 };

-static struct xt_table nat_table = {
+static struct xt_table __nat_table = {
  .name  = "nat",
  .valid_hooks = NAT_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
  .me  = THIS_MODULE,
  .af  = AF_INET,
 };
+static struct xt_table *nat_table;

 /* Source NAT */
 static unsigned int ipt_snat_target(struct sk_buff *skb,
@@ -214,7 +215,7 @@ int nf_nat_rule_find(struct sk_buff *skb,
 {
  int ret;

- ret = ipt_do_table(skb, hooknum, in, out, &nat_table);
+ ret = ipt_do_table(skb, hooknum, in, out, nat_table);

  if (ret == NF_ACCEPT) {
   if (!nf_nat_initialized(ct, HOOK2MANIP(hooknum)))
@@ -248,9 +249,10 @@ int __init nf_nat_rule_init(void)
 {
  int ret;

- ret = ipt_register_table(&nat_table, &nat_initial_table.repl);
- if (ret != 0)
```

```
-  return ret;
+ nat_table = ipt_register_table(&init_net, &__nat_table,
+           &nat_initial_table.repl);
+ if (IS_ERR(nat_table))
+  return PTR_ERR(nat_table);
  ret = xt_register_target(&ipt_snat_reg);
  if (ret != 0)
   goto unregister_table;
@@ -264,7 +266,7 @@ int __init nf_nat_rule_init(void)
  unregister_snat:
  xt_unregister_target(&ipt_snat_reg);
  unregister_table:
- ipt_unregister_table(&nat_table);
+ ipt_unregister_table(nat_table);

  return ret;
 }
@@ -273,5 +275,5 @@ void nf_nat_rule_cleanup(void)
 {
  xt_unregister_target(&ipt_dnat_reg);
  xt_unregister_target(&ipt_snat_reg);
- ipt_unregister_table(&nat_table);
+ ipt_unregister_table(nat_table);
 }
--- a/net/ipv6/netfilter/ip6_tables.c
+++ b/net/ipv6/netfilter/ip6_tables.c
@@ -2074,7 +2074,7 @@ do_ip6t_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
  return ret;
 }

-int ip6t_register_table(struct xt_table *table, const struct ip6t_replace *repl)
+struct xt_table *ip6t_register_table(struct xt_table *table, const struct ip6t_replace *repl)
 {
  int ret;
  struct xt_table_info *newinfo;
@@ -2084,8 +2084,10 @@ int ip6t_register_table(struct xt_table *table, const struct ip6t_replace
*repl)
  struct xt_table *new_table;

  newinfo = xt_alloc_table_info(repl->size);
- if (!newinfo)
-  return -ENOMEM;
+ if (!newinfo) {
+  ret = -ENOMEM;
+  goto out;
+ }

  /* choose the copy on our node/cpu, but dont care about preemption */
```

```
  loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
@@ -2096,18 +2098,20 @@ int ip6t_register_table(struct xt_table *table, const struct
ip6t_replace *repl)
          repl->num_entries,
          repl->hook_entry,
          repl->underflow);
- if (ret != 0) {
-  xt_free_table_info(newinfo);
-  return ret;
- }
+ if (ret != 0)
+  goto out_free;

  new_table = xt_register_table(&init_net, table, &bootstrap, newinfo);
  if (IS_ERR(new_table)) {
-  xt_free_table_info(newinfo);
-  return PTR_ERR(new_table);
+  ret = PTR_ERR(new_table);
+  goto out_free;
  }
+ return new_table;

- return 0;
+out_free:
+ xt_free_table_info(newinfo);
+out:
+ return ERR_PTR(ret);
 }

 void ip6t_unregister_table(struct xt_table *table)
--- a/net/ipv6/netfilter/ip6table_filter.c
+++ b/net/ipv6/netfilter/ip6table_filter.c
@@ -51,13 +51,14 @@ static struct
  .term = IP6T_ERROR_INIT,  /* ERROR */
 };

-static struct xt_table packet_filter = {
+static struct xt_table __packet_filter = {
  .name  = "filter",
  .valid_hooks = FILTER_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
  .me  = THIS_MODULE,
  .af  = AF_INET6,
 };
+static struct xt_table *packet_filter;

 /* The work comes in here from netfilter.c. */
 static unsigned int
```

```
@@ -67,7 +68,7 @@ ip6t_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ip6t_do_table(skb, hook, in, out, &packet_filter);
+ return ip6t_do_table(skb, hook, in, out, packet_filter);
 }

 static unsigned int
@@ -87,7 +88,7 @@ ip6t_local_out_hook(unsigned int hook,
  }
 #endif

- return ip6t_do_table(skb, hook, in, out, &packet_filter);
+ return ip6t_do_table(skb, hook, in, out, packet_filter);
 }

 static struct nf_hook_ops ip6t_ops[] __read_mostly = {
@@ -131,9 +132,9 @@ static int __init ip6table_filter_init(void)
  initial_table.entries[1].target.verdict = -forward - 1;

  /* Register table */
- ret = ip6t_register_table(&packet_filter, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_filter = ip6t_register_table(&__packet_filter, &initial_table.repl);
+ if (IS_ERR(packet_filter))
+  return PTR_ERR(packet_filter);

  /* Register hooks */
  ret = nf_register_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
@@ -143,14 +144,14 @@ static int __init ip6table_filter_init(void)
  return ret;

  cleanup_table:
- ip6t_unregister_table(&packet_filter);
+ ip6t_unregister_table(packet_filter);
  return ret;
 }

 static void __exit ip6table_filter_fini(void)
 {
  nf_unregister_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
- ip6t_unregister_table(&packet_filter);
+ ip6t_unregister_table(packet_filter);
 }

 module_init(ip6table_filter_init);
```

```
--- a/net/ipv6/netfilter/ip6table_mangle.c
+++ b/net/ipv6/netfilter/ip6table_mangle.c
@@ -57,13 +57,14 @@ static struct
  .term = IP6T_ERROR_INIT,  /* ERROR */
 };

-static struct xt_table packet_mangler = {
+static struct xt_table __packet_mangler = {
  .name  = "mangle",
  .valid_hooks = MANGLE_VALID_HOOKS,
  .lock  = RW_LOCK_UNLOCKED,
  .me  = THIS_MODULE,
  .af  = AF_INET6,
 };
+static struct xt_table *packet_mangler;

 /* The work comes in here from netfilter.c. */
 static unsigned int
@@ -73,7 +74,7 @@ ip6t_route_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ip6t_do_table(skb, hook, in, out, &packet_mangler);
+ return ip6t_do_table(skb, hook, in, out, packet_mangler);
 }

 static unsigned int
@@ -108,7 +109,7 @@ ip6t_local_hook(unsigned int hook,
  /* flowlabel and prio (includes version, which shouldn't change either */
  flowlabel = *((u_int32_t *)ipv6_hdr(skb));

- ret = ip6t_do_table(skb, hook, in, out, &packet_mangler);
+ ret = ip6t_do_table(skb, hook, in, out, packet_mangler);

  if (ret != NF_DROP && ret != NF_STOLEN
   && (memcmp(&ipv6_hdr(skb)->saddr, &saddr, sizeof(saddr))
@@ -163,9 +164,9 @@ static int __init ip6table_mangle_init(void)
  int ret;

  /* Register table */
- ret = ip6t_register_table(&packet_mangler, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_mangler = ip6t_register_table(&__packet_mangler, &initial_table.repl);
+ if (IS_ERR(packet_mangler))
+  return PTR_ERR(packet_mangler);

  /* Register hooks */
```

```
  ret = nf_register_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
@@ -175,14 +176,14 @@ static int __init ip6table_mangle_init(void)
  return ret;

  cleanup_table:
- ip6t_unregister_table(&packet_mangler);
+ ip6t_unregister_table(packet_mangler);
  return ret;
 }

 static void __exit ip6table_mangle_fini(void)
 {
  nf_unregister_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
- ip6t_unregister_table(&packet_mangler);
+ ip6t_unregister_table(packet_mangler);
 }

 module_init(ip6table_mangle_init);
--- a/net/ipv6/netfilter/ip6table_raw.c
+++ b/net/ipv6/netfilter/ip6table_raw.c
@@ -35,13 +35,14 @@ static struct
  .term = IP6T_ERROR_INIT,  /* ERROR */
 };

-static struct xt_table packet_raw = {
+static struct xt_table __packet_raw = {
  .name = "raw",
  .valid_hooks = RAW_VALID_HOOKS,
  .lock = RW_LOCK_UNLOCKED,
  .me = THIS_MODULE,
  .af = AF_INET6,
 };
+static struct xt_table *packet_raw;

 /* The work comes in here from netfilter.c. */
 static unsigned int
@@ -51,7 +52,7 @@ ip6t_hook(unsigned int hook,
   const struct net_device *out,
   int (*okfn)(struct sk_buff *))
 {
- return ip6t_do_table(skb, hook, in, out, &packet_raw);
+ return ip6t_do_table(skb, hook, in, out, packet_raw);
 }

 static struct nf_hook_ops ip6t_ops[] __read_mostly = {
@@ -76,9 +77,9 @@ static int __init ip6table_raw_init(void)
  int ret;
```

```
  /* Register table */
- ret = ip6t_register_table(&packet_raw, &initial_table.repl);
- if (ret < 0)
-  return ret;
+ packet_raw = ip6t_register_table(&__packet_raw, &initial_table.repl);
+ if (IS_ERR(packet_raw))
+  return PTR_ERR(packet_raw);

  /* Register hooks */
  ret = nf_register_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
@@ -88,14 +89,14 @@ static int __init ip6table_raw_init(void)
  return ret;

  cleanup_table:
- ip6t_unregister_table(&packet_raw);
+ ip6t_unregister_table(packet_raw);
  return ret;
 }

 static void __exit ip6table_raw_fini(void)
 {
  nf_unregister_hooks(ip6t_ops, ARRAY_SIZE(ip6t_ops));
- ip6t_unregister_table(&packet_raw);
+ ip6t_unregister_table(packet_raw);
 }

 module_init(ip6table_raw_init);
--- a/net/netfilter/x_tables.c
+++ b/net/netfilter/x_tables.c
@@ -667,9 +667,16 @@ struct xt_table *xt_register_table(struct net *net, struct xt_table *table,
  struct xt_table_info *private;
  struct xt_table *t;

+ /* Don't add one object to multiple lists. */
+ table = kmemdup(table, sizeof(struct xt_table), GFP_KERNEL);
+ if (!table) {
+  ret = -ENOMEM;
+  goto out;
+ }
+
  ret = mutex_lock_interruptible(&xt[table->af].mutex);
  if (ret != 0)
-  goto out;
+  goto out_free;

  /* Don't autoload: we'd eat our tail... */
  list_for_each_entry(t, &net->xt.tables[table->af], list) {
@@ -697,6 +704,8 @@ struct xt_table *xt_register_table(struct net *net, struct xt_table *table,
```

```
  unlock:
  mutex_unlock(&xt[table->af].mutex);
+out_free:
+ kfree(table);
 out:
  return ERR_PTR(ret);
 }
@@ -710,6 +719,7 @@ void *xt_unregister_table(struct xt_table *table)
  private = table->private;
  list_del(&table->list);
  mutex_unlock(&xt[table->af].mutex);
+ kfree(table);

  return private;
 }
```

---

Subject: Re: [PATCH 3/5] netns netfilter: return new table from {arp, ip, ip6}t_register_table()
Posted by Patrick McHardy on Tue, 22 Jan 2008 16:57:16 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> Typical table module registers xt_table structure (i.e. packet_filter)
> and link it to list during it. We can't use one template for it because
> corresponding list_head will become corrupted. We also can't unregister
> with template because it wasn't changed at all and thus doesn't know in
> which list it is.
>
> So, we duplicate template at the very first step of table registration.
> Table modules will save it for use during unregistration time and actual
> filtering.
>
> Do it at once to not screw bisection.

Applied, thanks.

> P.S.: renaming i.e. packet_filter => __packet_filter is temporary until
>       full netnsization of table modules is done.

It seems this could have been avoided by ordering the patches
differently (I probably would also have done 4/5 as 1/5).

---