Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Mishin Dmitry on Wed, 29 Mar 2006 11:36:16 GMT
View Forum Message <> Reply to Message

On Wednesday 29 March 2006 13:28, Patrick McHardy wrote:
> Dmitry Mishin wrote:
> > This patch extends current iptables compatibility layer in order to get
> > 32bit iptables to work on 64bit kernel. Current layer is insufficient due
> > to alignment checks both in kernel and user space tools.
> >
> > Patch is for current net-2.6.17 with addition of move of
> > ipt_entry_{match| target} definitions to xt_entry_{match|target}.
>
> Thanks, this looks good. Two small issues so far:
> > diff --git a/net/compat.c b/net/compat.c
> > index 13177a1..6a7028e 100644
> > --- a/net/compat.c
> > +++ b/net/compat.c
> > @@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
> > int err;
> > struct socket *sock;
> >
> > - /* SO_SET_REPLACE seems to be the same in all levels */
> > - if (optname == IPT_SO_SET_REPLACE)
> > + if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
> > return do_netfilter_replace(fd, level, optname,
> > optval, optlen);
>
> I don't understand the reason for this change. If its not a mistake,
> it would make more sense to check for IP6T_SO_SET_REPLACE I guess ..
IP6T_SO_SET_REPLACE == IPT_SO_SET_REPLACE == XT_SO_SET_REPLACE.
Rename will require respective #include directive rename, so, I just leave
this as it is. BTW, I'll make respective patch for IPV6 in the near future
and this hunk will be removed at all.

>
> > +#ifdef CONFIG_COMPAT
> > +void xt_compat_lock(int af)
> > +{
> > + down(&xt[af].compat_mutex);
> > +}
> > +EXPORT_SYMBOL_GPL(xt_compat_lock);
> > +
> > +void xt_compat_unlock(int af)
> > +{
> > + up(&xt[af].compat_mutex);
> > +}
> > +EXPORT_SYMBOL_GPL(xt_compat_unlock);

> > +#endif
>
> Won't a seperate compat-mutex introduce races between compat- and
> non-compat users? BTW, the up/down calls have been replaced by the
> new mutex API in Linus' tree, please resend the patch against the
> current tree.
compat_mutex is always over xt[af].mutex and can't be taken under the last
one, so, there should be no races.
New patch is attached.

--
Thanks,
Dmitry.

```
diff --git a/include/linux/netfilter/x_tables.h b/include/linux/netfilter/x_tables.h
index 1350e47..f6bdef8 100644
--- a/include/linux/netfilter/x_tables.h
+++ b/include/linux/netfilter/x_tables.h
@@ -142,6 +142,12 @@ struct xt_counters_info
 #define ASSERT_WRITE_LOCK(x)
 #include <linux/netfilter_ipv4/listhelp.h>

+#ifdef CONFIG_COMPAT
+#define COMPAT_TO_USER  1
+#define COMPAT_FROM_USER -1
+#define COMPAT_CALC_SIZE 0
+#endif
+
 struct xt_match
 {
  struct list_head list;
@@ -175,6 +181,9 @@ struct xt_match
  void (*destroy)(const struct xt_match *match, void *matchinfo,
    unsigned int matchinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *match, void **dstptr, int *size, int convert);
+
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -220,6 +229,9 @@ struct xt_target
  void (*destroy)(const struct xt_target *target, void *targinfo,
    unsigned int targinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *target, void **dstptr, int *size, int convert);
+
```

```
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -314,6 +326,61 @@ extern void xt_proto_fini(int af);
 extern struct xt_table_info *xt_alloc_table_info(unsigned int size);
 extern void xt_free_table_info(struct xt_table_info *info);

+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_xt_entry_match
+{
+ union {
+  struct {
+   u_int16_t match_size;
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
+  u_int16_t match_size;
+ } u;
+ unsigned char data[0];
+};
+
+struct compat_xt_entry_target
+{
+ union {
+  struct {
+   u_int16_t target_size;
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
+  u_int16_t target_size;
+ } u;
+ unsigned char data[0];
+};
+
+/* FIXME: this works only on 32 bit tasks
+ * need to change whole approach in order to calculate align as function of
+ * current task alignment */
+
+struct compat_xt_counters
+{
+ u_int32_t cnt[4];
+};
+
+struct compat_xt_counters_info
+{
+ char name[XT_TABLE_MAXNAMELEN];
```

```
+ compat_uint_t num_counters;
+ struct compat_xt_counters counters[0];
+};
+
+#define COMPAT_XT_ALIGN(s) (((s) + (__alignof__(struct compat_xt_counters)-1)) \
+  & ~(__alignof__(struct compat_xt_counters)-1))
+
+extern void xt_compat_lock(int af);
+extern void xt_compat_unlock(int af);
+extern int xt_compat_match(void *match, void **dstptr, int *size, int convert);
+extern int xt_compat_target(void *target, void **dstptr, int *size,
+  int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /* __KERNEL__ */

 #endif /* _X_TABLES_H */
diff --git a/include/linux/netfilter_ipv4/ip_tables.h b/include/linux/netfilter_ipv4/ip_tables.h
index d5b8c0d..c0dac16 100644
--- a/include/linux/netfilter_ipv4/ip_tables.h
+++ b/include/linux/netfilter_ipv4/ip_tables.h
@@ -316,5 +316,23 @@ extern unsigned int ipt_do_table(struct
      void *userdata);

 #define IPT_ALIGN(s) XT_ALIGN(s)
+
+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_ipt_entry
+{
+ struct ipt_ip ip;
+ compat_uint_t nfcache;
+ u_int16_t target_offset;
+ u_int16_t next_offset;
+ compat_uint_t comefrom;
+ struct compat_xt_counters counters;
+ unsigned char elems[0];
+};
+
+#define COMPAT_IPT_ALIGN(s)  COMPAT_XT_ALIGN(s)
+
+#endif /* CONFIG_COMPAT */
 #endif /*__KERNEL__*/
 #endif /* _IPTABLES_H */
diff --git a/net/compat.c b/net/compat.c
index 8fd37cd..d5d69fa 100644
--- a/net/compat.c
```

```
+++ b/net/compat.c
@@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
  int err;
  struct socket *sock;

- /* SO_SET_REPLACE seems to be the same in all levels */
- if (optname == IPT_SO_SET_REPLACE)
+ if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);

diff --git a/net/ipv4/netfilter/ip_tables.c b/net/ipv4/netfilter/ip_tables.c
index a7b194c..34df287 100644
--- a/net/ipv4/netfilter/ip_tables.c
+++ b/net/ipv4/netfilter/ip_tables.c
@@ -24,6 +24,7 @@
 #include <linux/module.h>
 #include <linux/icmp.h>
 #include <net/ip.h>
+#include <net/compat.h>
 #include <asm/uaccess.h>
 #include <linux/mutex.h>
 #include <linux/proc_fs.h>
@@ -799,17 +800,11 @@ get_counters(const struct xt_table_info
  }
 }

-static int
-copy_entries_to_user(unsigned int total_size,
-       struct ipt_table *table,
-       void __user *userptr)
+static inline struct xt_counters * alloc_counters(struct ipt_table *table)
 {
- unsigned int off, num, countersize;
- struct ipt_entry *e;
+ unsigned int countersize;
  struct xt_counters *counters;
  struct xt_table_info *private = table->private;
- int ret = 0;
- void *loc_cpu_entry;

  /* We need atomic snapshot of counters: rest doesn't change
     (other than comefrom, which userspace doesn't care
@@ -818,13 +813,32 @@ copy_entries_to_user(unsigned int total_
  counters = vmalloc_node(countersize, numa_node_id());

  if (counters == NULL)
-  return -ENOMEM;
```

```
+  return ERR_PTR(-ENOMEM);

  /* First, sum counters... */
  write_lock_bh(&table->lock);
  get_counters(private, counters);
  write_unlock_bh(&table->lock);

+ return counters;
+}
+
+static int
+copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table,
+       void __user *userptr)
+{
+ unsigned int off, num;
+ struct ipt_entry *e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ int ret = 0;
+ void *loc_cpu_entry;
+
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
  /* choose the copy that is on our node/cpu, ...
   * This choice is lazy (because current thread is
   * allowed to migrate to another cpu)
@@ -878,50 +892,905 @@ copy_entries_to_user(unsigned int total_
    goto free_counters;
   }
  }
-
- free_counters:
- vfree(counters);
+
+ free_counters:
+ vfree(counters);
+ return ret;
+}
+
+#ifdef CONFIG_COMPAT
+struct compat_delta {
+ struct compat_delta *next;
+ u_int16_t offset;
+ short delta;
+};
```

```
+
+static struct compat_delta *compat_offsets = NULL;
+
+static int compat_add_offset(u_int16_t offset, short delta)
+{
+ struct compat_delta *tmp;
+
+ tmp = kmalloc(sizeof(struct compat_delta), GFP_KERNEL);
+ if (!tmp)
+  return -ENOMEM;
+ tmp->offset = offset;
+ tmp->delta = delta;
+ if (compat_offsets) {
+  tmp->next = compat_offsets->next;
+  compat_offsets->next = tmp;
+ } else {
+  compat_offsets = tmp;
+  tmp->next = NULL;
+ }
+ return 0;
+}
+
+static void compat_flush_offsets(void)
+{
+ struct compat_delta *tmp, *next;
+
+ if (compat_offsets) {
+  for(tmp = compat_offsets; tmp; tmp = next) {
+   next = tmp->next;
+   kfree(tmp);
+  }
+  compat_offsets = NULL;
+ }
+}
+
+static short compat_calc_jump(u_int16_t offset)
+{
+ struct compat_delta *tmp;
+ short delta;
+
+ for(tmp = compat_offsets, delta = 0; tmp; tmp = tmp->next)
+  if (tmp->offset < offset)
+   delta += tmp->delta;
+ return delta;
+}
+
+struct compat_ipt_standard_target
+{
```

```
+ struct compat_xt_entry_target target;
+ compat_int_t verdict;
+};
+
+#define IPT_ST_OFFSET (sizeof(struct ipt_standard_target) - \
+    sizeof(struct compat_ipt_standard_target))
+
+struct compat_ipt_standard
+{
+ struct compat_ipt_entry entry;
+ struct compat_ipt_standard_target target;
+};
+
+static int compat_ipt_standard_fn(void *target,
+  void **dstptr, int *size, int convert)
+{
+ struct compat_ipt_standard_target compat_st, *pcompat_st;
+ struct ipt_standard_target st, *pst;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pst = (struct ipt_standard_target *)target;
+   memcpy(&compat_st.target, &pst->target,
+     sizeof(struct ipt_entry_target));
+   compat_st.verdict = pst->verdict;
+   if (compat_st.verdict > 0)
+    compat_st.verdict -=
+     compat_calc_jump(compat_st.verdict);
+   compat_st.target.u.user.target_size =
+   sizeof(struct compat_ipt_standard_target);
+   if (__copy_to_user(*dstptr, &compat_st,
+    sizeof(struct compat_ipt_standard_target)))
+    ret = -EFAULT;
+   *size -= IPT_ST_OFFSET;
+   *dstptr += sizeof(struct compat_ipt_standard_target);
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_st =
+    (struct compat_ipt_standard_target *)target;
+   memcpy(&st.target, &pcompat_st->target,
+     sizeof(struct ipt_entry_target));
+   st.verdict = pcompat_st->verdict;
+   if (st.verdict > 0)
+    st.verdict += compat_calc_jump(st.verdict);
+   st.target.u.user.target_size =
+   sizeof(struct ipt_standard_target);
```

```
+   memcpy(*dstptr, &st,
+     sizeof(struct ipt_standard_target));
+   *size += IPT_ST_OFFSET;
+   *dstptr += sizeof(struct ipt_standard_target);
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += IPT_ST_OFFSET;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+static inline int
+compat_calc_match(struct ipt_entry_match *m, int * size)
+{
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+ return 0;
+}
+
+static int compat_calc_entry(struct ipt_entry *e, struct xt_table_info *info,
+  void *base, struct xt_table_info *newinfo)
+{
+ struct ipt_entry_target *t;
+ u_int16_t entry_offset;
+ int off, i, ret;
+
+ off = 0;
+ entry_offset = (void *)e - base;
+ IPT_MATCH_ITERATE(e, compat_calc_match, &off);
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ newinfo->size -= off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  return ret;
+
+ for (i = 0; i< NF_IP_NUMHOOKS; i++) {
+  if (info->hook_entry[i] && (e < (struct ipt_entry *)
+    (base + info->hook_entry[i])))
```

```
+   newinfo->hook_entry[i] -= off;
+  if (info->underflow[i] && (e < (struct ipt_entry *)
+    (base + info->underflow[i])))
+   newinfo->underflow[i] -= off;
+ }
+ return 0;
+}
+
+static int compat_table_info(struct xt_table_info *info,
+  struct xt_table_info *newinfo)
+{
+ void *loc_cpu_entry;
+ int i;
+
+ if (!newinfo || !info)
+  return -EINVAL;
+
+ memset(newinfo, 0, sizeof(struct xt_table_info));
+ newinfo->size = info->size;
+ newinfo->number = info->number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ loc_cpu_entry = info->entries[raw_smp_processor_id()];
+ return IPT_ENTRY_ITERATE(loc_cpu_entry, info->size,
+   compat_calc_entry, info, loc_cpu_entry, newinfo);
+}
+#endif
+
+static int get_info(void __user *user, int *len, int compat)
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ struct ipt_table *t;
+ int ret;
+
+ if (*len != sizeof(struct ipt_getinfo)) {
+  duprintf("length %u != %u\n", *len,
+   (unsigned int)sizeof(struct ipt_getinfo));
+  return -EINVAL;
+ }
+
+ if (copy_from_user(name, user, sizeof(name)) != 0)
+  return -EFAULT;
+
+ name[IPT_TABLE_MAXNAMELEN-1] = '\0';
+#ifdef CONFIG_COMPAT
+ if (compat)
```

```
+  xt_compat_lock(AF_INET);
+#endif
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+   "iptable_%s", name);
+ if (t && !IS_ERR(t)) {
+  struct ipt_getinfo info;
+  struct xt_table_info *private = t->private;
+
+#ifdef CONFIG_COMPAT
+  if (compat) {
+   struct xt_table_info tmp;
+   ret = compat_table_info(private, &tmp);
+   compat_flush_offsets();
+   private =  &tmp;
+  }
+#endif
+  info.valid_hooks = t->valid_hooks;
+  memcpy(info.hook_entry, private->hook_entry,
+    sizeof(info.hook_entry));
+  memcpy(info.underflow, private->underflow,
+    sizeof(info.underflow));
+  info.num_entries = private->number;
+  info.size = private->size;
+  strcpy(info.name, name);
+
+  if (copy_to_user(user, &info, *len) != 0)
+   ret = -EFAULT;
+  else
+   ret = 0;
+
+  xt_table_unlock(t);
+  module_put(t->me);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+#ifdef CONFIG_COMPAT
+ if (compat)
+  xt_compat_unlock(AF_INET);
+#endif
+ return ret;
+}
+
+static int
+get_entries(struct ipt_get_entries __user *uptr, int *len)
+{
+ int ret;
+ struct ipt_get_entries get;
+ struct ipt_table *t;
+
```

```
+ if (*len < sizeof(get)) {
+  duprintf("get_entries: %u < %d\n", *len,
+    (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+ if (*len != sizeof(struct ipt_get_entries) + get.size) {
+  duprintf("get_entries: %u != %u\n", *len,
+    (unsigned int)(sizeof(struct ipt_get_entries) +
+    get.size));
+  return -EINVAL;
+ }
+
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  if (get.size == private->size)
+   ret = copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  else {
+   duprintf("get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+
+ return ret;
+}
+
+static int
+__do_replace(const char *name, unsigned int valid_hooks,
+  struct xt_table_info *newinfo, unsigned int num_counters,
+  void __user *counters_ptr)
+{
+ int ret;
+ struct ipt_table *t;
+ struct xt_table_info *oldinfo;
+ struct xt_counters *counters;
+ void *loc_cpu_old_entry;
+
+ ret = 0;
```

```
+ counters = vmalloc(num_counters * sizeof(struct xt_counters));
+ if (!counters) {
+  ret = -ENOMEM;
+  goto out;
+ }
+
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+        "iptable_%s", name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free_newinfo_counters_untrans;
+ }
+
+ /* You lied! */
+ if (valid_hooks != t->valid_hooks) {
+  duprintf("Valid hook crap: %08X vs %08X\n",
+    valid_hooks, t->valid_hooks);
+  ret = -EINVAL;
+  goto put_module;
+ }
+
+ oldinfo = xt_replace_table(t, num_counters, newinfo, &ret);
+ if (!oldinfo)
+  goto put_module;
+
+ /* Update module usage count based on number of rules */
+ duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
+  oldinfo->number, oldinfo->initial_entries, newinfo->number);
+ if ((oldinfo->number > oldinfo->initial_entries) ||
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
+ if ((oldinfo->number > oldinfo->initial_entries) &&
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
+
+ /* Get the old counters. */
+ get_counters(oldinfo, counters);
+ /* Decrease module usage counts and free resource */
+ loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
+ xt_free_table_info(oldinfo);
+ if (copy_to_user(counters_ptr, counters,
+    sizeof(struct xt_counters) * num_counters) != 0)
+  ret = -EFAULT;
+ vfree(counters);
+ xt_table_unlock(t);
+ return ret;
+
```

```
+ put_module:
+ module_put(t->me);
+ xt_table_unlock(t);
+ free_newinfo_counters_untrans:
+ vfree(counters);
+ out:
+ return ret;
+}
+
+static int
+do_replace(void __user *user, unsigned int len)
+{
+ int ret;
+ struct ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;
+
+ if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+  return -EFAULT;
+
+ /* Hack: Causes ipchains to give correct error msg --RR */
+ if (len != sizeof(tmp) + tmp.size)
+  return -ENOPROTOOPT;
+
+ /* overflow check */
+ if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
+   SMP_CACHE_BYTES)
+  return -ENOMEM;
+ if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
+  return -ENOMEM;
+
+ newinfo = xt_alloc_table_info(tmp.size);
+ if (!newinfo)
+  return -ENOMEM;
+
+ /* choose the copy that is our node/cpu */
+ loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
+ if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
+      tmp.size) != 0) {
+  ret = -EFAULT;
+  goto free_newinfo;
+ }
+
+ ret = translate_table(tmp.name, tmp.valid_hooks,
+         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
+         tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
+  goto free_newinfo;
```

```
+
+ duprintf("ip_tables: Translated table\n");
+
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         tmp.counters);
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}
+
+/* We're lazy, and add to the first CPU; overflow works its fey magic
+ * and everything is OK. */
+static inline int
+add_counter_to_entry(struct ipt_entry *e,
+       const struct xt_counters addme[],
+       unsigned int *i)
+{
+#if 0
+ duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
+    *i,
+    (long unsigned int)e->counters.pcnt,
+    (long unsigned int)e->counters.bcnt,
+    (long unsigned int)addme[*i].pcnt,
+    (long unsigned int)addme[*i].bcnt);
+#endif
+
+ ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+
+ (*i)++;
+ return 0;
+}
+
+static int
+do_add_counters(void __user *user, unsigned int len, int compat)
+{
+ unsigned int i;
+ struct xt_counters_info tmp;
+ struct xt_counters *paddc;
+ unsigned int num_counters;
+ char *name;
+ int size;
```

```
+ void *ptmp;
+ struct ipt_table *t;
+ struct xt_table_info *private;
+ int ret = 0;
+ void *loc_cpu_entry;
+#ifdef CONFIG_COMPAT
+ struct compat_xt_counters_info compat_tmp;
+
+ if (compat) {
+  ptmp = &compat_tmp;
+  size = sizeof(struct compat_xt_counters_info);
+ } else
+#endif
+ {
+  ptmp = &tmp;
+  size = sizeof(struct xt_counters_info);
+ }
+
+ if (copy_from_user(ptmp, user, size) != 0)
+  return -EFAULT;
+
+#ifdef CONFIG_COMPAT
+ if (compat) {
+  num_counters = compat_tmp.num_counters;
+  name = compat_tmp.name;
+ } else
+#endif
+ {
+  num_counters = tmp.num_counters;
+  name = tmp.name;
+ }
+
+ if (len != size + num_counters * sizeof(struct xt_counters))
+  return -EINVAL;
+
+ paddc = vmalloc_node(len - size, numa_node_id());
+ if (!paddc)
+  return -ENOMEM;
+
+ if (copy_from_user(paddc, user + size, len - size) != 0) {
+  ret = -EFAULT;
+  goto free;
+ }
+
+ t = xt_find_table_lock(AF_INET, name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free;
```

```
+ }
+
+ write_lock_bh(&t->lock);
+ private = t->private;
+ if (private->number != num_counters) {
+  ret = -EINVAL;
+  goto unlock_up_free;
+ }
+
+ i = 0;
+ /* Choose the copy that is on our node */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_entry,
+     private->size,
+     add_counter_to_entry,
+     paddc,
+     &i);
+ unlock_up_free:
+ write_unlock_bh(&t->lock);
+ xt_table_unlock(t);
+ module_put(t->me);
+ free:
+ vfree(paddc);
+
+ return ret;
+}
+
+#ifdef CONFIG_COMPAT
+struct compat_ipt_replace {
+ char   name[IPT_TABLE_MAXNAMELEN];
+ u32   valid_hooks;
+ u32   num_entries;
+ u32   size;
+ u32   hook_entry[NF_IP_NUMHOOKS];
+ u32   underflow[NF_IP_NUMHOOKS];
+ u32   num_counters;
+ compat_uptr_t  counters; /* struct ipt_counters * */
+ struct compat_ipt_entry entries[0];
+};
+
+static inline int compat_copy_match_to_user(struct ipt_entry_match *m,
+  void __user **dstptr, compat_uint_t *size)
+{
+ if (m->u.kernel.match->compat)
+  return m->u.kernel.match->compat(m, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  return xt_compat_match(m, dstptr, size, COMPAT_TO_USER);
```

```
+}
+
+static int compat_copy_entry_to_user(struct ipt_entry *e,
+  void __user **dstptr, compat_uint_t *size)
+{
+ struct ipt_entry_target __user *t;
+ struct compat_ipt_entry __user *ce;
+ u_int16_t target_offset, next_offset;
+ compat_uint_t origsize;
+ int ret;
+
+ ret = -EFAULT;
+ origsize = *size;
+ ce = (struct compat_ipt_entry __user *)*dstptr;
+ if (__copy_to_user(ce, e, sizeof(struct ipt_entry)))
+  goto out;
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_to_user, dstptr, size);
+ target_offset = e->target_offset - (origsize - *size);
+ if (ret)
+  goto out;
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  ret = t->u.kernel.target->compat(t, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  ret = xt_compat_target(t, dstptr, size, COMPAT_TO_USER);
+ if (ret)
+  goto out;
+ ret = -EFAULT;
+ next_offset = e->next_offset - (origsize - *size);
+ if (__put_user(target_offset, &ce->target_offset))
+  goto out;
+ if (__put_user(next_offset, &ce->next_offset))
+  goto out;
+ return 0;
+out:
+ return ret;
+}
+
+static inline int
+compat_check_calc_match(struct ipt_entry_match *m,
+     const char *name,
+     const struct ipt_ip *ip,
+     unsigned int hookmask,
+     int *size, int *i)
+{
```

```
+ struct ipt_match *match;
+
+ match = try_then_request_module(xt_find_match(AF_INET, m->u.user.name,
+         m->u.user.revision),
+     "ipt_%s", m->u.user.name);
+ if (IS_ERR(match) || !match) {
+  duprintf("compat_check_calc_match: `%s' not found\n",
+    m->u.user.name);
+  return match ? PTR_ERR(match) : -ENOENT;
+ }
+ m->u.kernel.match = match;
+
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+
+ (*i)++;
+ return 0;
+}
+
+static inline int
+check_compat_entry_size_and_hooks(struct ipt_entry *e,
+      struct xt_table_info *newinfo,
+      unsigned int *size,
+      unsigned char *base,
+      unsigned char *limit,
+      unsigned int *hook_entries,
+      unsigned int *underflows,
+      unsigned int *i,
+      const char *name)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ u_int16_t entry_offset;
+ int ret, off, h, j;
+
+ duprintf("check_compat_entry_size_and_hooks %p\n", e);
+ if ((unsigned long)e % __alignof__(struct compat_ipt_entry) != 0
+     || (unsigned char *)e + sizeof(struct compat_ipt_entry) >= limit) {
+  duprintf("Bad offset %p, limit = %p\n", e, limit);
+  return -EINVAL;
+ }
+
+ if (e->next_offset < sizeof(struct compat_ipt_entry) +
+   sizeof(struct compat_xt_entry_target)) {
+  duprintf("checking: element %p size %u\n",
+    e, e->next_offset);
```

```
+  return -EINVAL;
+ }
+
+ if (!ip_checkentry(&e->ip)) {
+  duprintf("ip_tables: ip check failed %p %s.\n", e, name);
+  return -EINVAL;
+ }
+
+ off = 0;
+ entry_offset = (void *)e - (void *)base;
+ j = 0;
+ ret = IPT_MATCH_ITERATE(e, compat_check_calc_match, name, &e->ip,
+   e->comefrom, &off, &j);
+ if (ret != 0)
+  goto out;
+
+ t = ipt_get_target(e);
+ target = try_then_request_module(xt_find_target(AF_INET,
+           t->u.user.name,
+           t->u.user.revision),
+      "ipt_%s", t->u.user.name);
+ if (IS_ERR(target) || !target) {
+  duprintf("check_entry: `%s' not found\n", t->u.user.name);
+  ret = target ? PTR_ERR(target) : -ENOENT;
+  goto out;
+ }
+ t->u.kernel.target = target;
+
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ *size += off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  goto out;
+
+ /* Check hooks & underflows */
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)e - base == hook_entries[h])
+   newinfo->hook_entry[h] = hook_entries[h];
+  if ((unsigned char *)e - base == underflows[h])
+   newinfo->underflow[h] = underflows[h];
+ }
+
+ /* Clear counters and comefrom */
+ e->counters = ((struct ipt_counters) { 0, 0 });
+ e->comefrom = 0;
```

```
+
+ (*i)++;
+ return 0;
+out:
+ IPT_MATCH_ITERATE(e, cleanup_match, &j);
+ return ret;
+}
+
+static inline int compat_copy_match_from_user(struct ipt_entry_match *m,
+ void **dstptr, compat_uint_t *size, const char *name,
+ const struct ipt_ip *ip, unsigned int hookmask)
+{
+ struct ipt_entry_match *dm;
+ struct ipt_match *match;
+ int ret;
+
+ dm = (struct ipt_entry_match *)*dstptr;
+ match = m->u.kernel.match;
+ if (match->compat)
+  match->compat(m, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_match(m, dstptr, size, COMPAT_FROM_USER);
+
+ ret = xt_check_match(match, AF_INET, dm->u.match_size - sizeof(*dm),
+        name, hookmask, ip->proto,
+        ip->invflags & IPT_INV_PROTO);
+ if (ret)
+  return ret;
+
+ if (m->u.kernel.match->checkentry
+     && !m->u.kernel.match->checkentry(name, ip, match, dm->data,
+           dm->u.match_size - sizeof(*dm),
+           hookmask)) {
+  duprintf("ip_tables: check failed for `%s'.\n",
+    m->u.kernel.match->name);
+  return -EINVAL;
+ }
+ return 0;
+}
+
+static int compat_copy_entry_from_user(struct ipt_entry *e, void **dstptr,
+ unsigned int *size, const char *name,
+ struct xt_table_info *newinfo, unsigned char *base)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ struct ipt_entry *de;
+ unsigned int origsize;
```

```
+ int ret, h;
+
+ ret = 0;
+ origsize = *size;
+ de = (struct ipt_entry *)*dstptr;
+ memcpy(de, e, sizeof(struct ipt_entry));
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_from_user, dstptr, size,
+   name, &de->ip, de->comefrom);
+ if (ret)
+  goto out;
+ de->target_offset = e->target_offset - (origsize - *size);
+ t = ipt_get_target(e);
+ target = t->u.kernel.target;
+ if (target->compat)
+  target->compat(t, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_target(t, dstptr, size, COMPAT_FROM_USER);
+
+ de->next_offset = e->next_offset - (origsize - *size);
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)de - base < newinfo->hook_entry[h])
+   newinfo->hook_entry[h] -= origsize - *size;
+  if ((unsigned char *)de - base < newinfo->underflow[h])
+   newinfo->underflow[h] -= origsize - *size;
+ }
+
+ t = ipt_get_target(de);
+ target = t->u.kernel.target;
+ ret = xt_check_target(target, AF_INET, t->u.target_size - sizeof(*t),
+         name, e->comefrom, e->ip.proto,
+         e->ip.invflags & IPT_INV_PROTO);
+ if (ret)
+  goto out;
+
+ ret = -EINVAL;
+ if (t->u.kernel.target == &ipt_standard_target) {
+  if (!standard_check(t, *size))
+   goto out;
+ } else if (t->u.kernel.target->checkentry
+     && !t->u.kernel.target->checkentry(name, de, target,
+    t->data, t->u.target_size - sizeof(*t),
+    de->comefrom)) {
+  duprintf("ip_tables: compat: check failed for `%s'.\n",
+    t->u.kernel.target->name);
+  goto out;
+ }
```

```
+ ret = 0;
+out:
  return ret;
 }

 static int
-get_entries(const struct ipt_get_entries *entries,
-     struct ipt_get_entries __user *uptr)
+translate_compat_table(const char *name,
+  unsigned int valid_hooks,
+  struct xt_table_info **pinfo,
+  void **pentry0,
+  unsigned int total_size,
+  unsigned int number,
+  unsigned int *hook_entries,
+  unsigned int *underflows)
 {
+ unsigned int i;
+ struct xt_table_info *newinfo, *info;
+ void *pos, *entry0, *entry1;
+ unsigned int size;
  int ret;
- struct ipt_table *t;

- t = xt_find_table_lock(AF_INET, entries->name);
- if (t && !IS_ERR(t)) {
-  struct xt_table_info *private = t->private;
-  duprintf("t->private->number = %u\n",
-    private->number);
-  if (entries->size == private->size)
-   ret = copy_entries_to_user(private->size,
-         t, uptr->entrytable);
-  else {
-   duprintf("get_entries: I've got %u not %u!\n",
-     private->size,
-     entries->size);
-   ret = -EINVAL;
+ info = *pinfo;
+ entry0 = *pentry0;
+ size = total_size;
+ info->number = number;
+
+ /* Init all hooks to impossible value. */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  info->hook_entry[i] = 0xFFFFFFFF;
+  info->underflow[i] = 0xFFFFFFFF;
+ }
+
```

```
+ duprintf("translate_compat_table: size %u\n", info->size);
+ i = 0;
+ xt_compat_lock(AF_INET);
+ /* Walk through entries, checking offsets. */
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+    check_compat_entry_size_and_hooks,
+    info, &size, entry0,
+    entry0 + total_size,
+    hook_entries, underflows, &i, name);
+ if (ret != 0)
+  goto out_unlock;
+
+ ret = -EINVAL;
+ if (i != number) {
+  duprintf("translate_compat_table: %u not %u entries\n",
+    i, number);
+  goto out_unlock;
+ }
+
+ /* Check hooks all assigned */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  /* Only hooks which are valid */
+  if (!(valid_hooks & (1 << i)))
+   continue;
+  if (info->hook_entry[i] == 0xFFFFFFFF) {
+   duprintf("Invalid hook entry %u %u\n",
+     i, hook_entries[i]);
+   goto out_unlock;
   }
-  module_put(t->me);
-  xt_table_unlock(t);
- } else
-  ret = t ? PTR_ERR(t) : -ENOENT;
+  if (info->underflow[i] == 0xFFFFFFFF) {
+   duprintf("Invalid underflow %u %u\n",
+     i, underflows[i]);
+   goto out_unlock;
+  }
+ }
+
+ ret = -ENOMEM;
+ newinfo = xt_alloc_table_info(size);
+ if (!newinfo)
+  goto out_unlock;
+
+ newinfo->number = number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
```

```
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ entry1 = newinfo->entries[raw_smp_processor_id()];
+ pos = entry1;
+ size =  total_size;
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+   compat_copy_entry_from_user, &pos, &size,
+   name, newinfo, entry1);
+ compat_flush_offsets();
+ xt_compat_unlock(AF_INET);
+ if (ret)
+  goto free_newinfo;
+
+ ret = -ELOOP;
+ if (!mark_source_chains(newinfo, valid_hooks, entry1))
+  goto free_newinfo;
+
+ /* And one copy for every other CPU */
+ for_each_cpu(i)
+  if (newinfo->entries[i] && newinfo->entries[i] != entry1)
+   memcpy(newinfo->entries[i], entry1, newinfo->size);
+
+ *pinfo = newinfo;
+ *pentry0 = entry1;
+ xt_free_table_info(info);
+ return 0;

+free_newinfo:
+ xt_free_table_info(newinfo);
+out:
  return ret;
+out_unlock:
+ xt_compat_unlock(AF_INET);
+ goto out;
 }

 static int
-do_replace(void __user *user, unsigned int len)
+compat_do_replace(void __user *user, unsigned int len)
 {
  int ret;
- struct ipt_replace tmp;
- struct ipt_table *t;
- struct xt_table_info *newinfo, *oldinfo;
- struct xt_counters *counters;
- void *loc_cpu_entry, *loc_cpu_old_entry;
+ struct compat_ipt_replace tmp;
+ struct xt_table_info *newinfo;
```

```
+ void *loc_cpu_entry;

  if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
   return -EFAULT;
@@ -949,151 +1818,201 @@ do_replace(void __user *user, unsigned i
   goto free_newinfo;
  }

- counters = vmalloc(tmp.num_counters * sizeof(struct xt_counters));
- if (!counters) {
-  ret = -ENOMEM;
+ ret = translate_compat_table(tmp.name, tmp.valid_hooks,
+         &newinfo, &loc_cpu_entry, tmp.size,
+         tmp.num_entries, tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
   goto free_newinfo;
- }

- ret = translate_table(tmp.name, tmp.valid_hooks,
-         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
-         tmp.hook_entry, tmp.underflow);
- if (ret != 0)
-  goto free_newinfo_counters;
+ duprintf("compat_do_replace: Translated table\n");

- duprintf("ip_tables: Translated table\n");
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         compat_ptr(tmp.counters));
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;

- t = try_then_request_module(xt_find_table_lock(AF_INET, tmp.name),
-        "iptable_%s", tmp.name);
- if (!t || IS_ERR(t)) {
-  ret = t ? PTR_ERR(t) : -ENOENT;
-  goto free_newinfo_counters_untrans;
- }
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}

- /* You lied! */
- if (tmp.valid_hooks != t->valid_hooks) {
```

```
-  duprintf("Valid hook crap: %08X vs %08X\n",
-    tmp.valid_hooks, t->valid_hooks);
-  ret = -EINVAL;
-  goto put_module;
- }
+static int
+compat_do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user,
+  unsigned int len)
+{
+ int ret;

- oldinfo = xt_replace_table(t, tmp.num_counters, newinfo, &ret);
- if (!oldinfo)
-  goto put_module;
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;

- /* Update module usage count based on number of rules */
- duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
-  oldinfo->number, oldinfo->initial_entries, newinfo->number);
- if ((oldinfo->number > oldinfo->initial_entries) ||
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
- if ((oldinfo->number > oldinfo->initial_entries) &&
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
+ switch (cmd) {
+ case IPT_SO_SET_REPLACE:
+  ret = compat_do_replace(user, len);
+  break;

- /* Get the old counters. */
- get_counters(oldinfo, counters);
- /* Decrease module usage counts and free resource */
- loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
- xt_free_table_info(oldinfo);
- if (copy_to_user(tmp.counters, counters,
-    sizeof(struct xt_counters) * tmp.num_counters) != 0)
-  ret = -EFAULT;
- vfree(counters);
- xt_table_unlock(t);
- return ret;
+ case IPT_SO_SET_ADD_COUNTERS:
+  ret = do_add_counters(user, len, 1);
+  break;
+
+ default:
```

```
+  duprintf("do_ipt_set_ctl:  unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }

- put_module:
- module_put(t->me);
- xt_table_unlock(t);
- free_newinfo_counters_untrans:
- IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
- free_newinfo_counters:
- vfree(counters);
- free_newinfo:
- xt_free_table_info(newinfo);
  return ret;
 }

-/* We're lazy, and add to the first CPU; overflow works its fey magic
- * and everything is OK. */
-static inline int
-add_counter_to_entry(struct ipt_entry *e,
-       const struct xt_counters addme[],
-       unsigned int *i)
+struct compat_ipt_get_entries
 {
-#if 0
- duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
-   *i,
-   (long unsigned int)e->counters.pcnt,
-   (long unsigned int)e->counters.bcnt,
-   (long unsigned int)addme[*i].pcnt,
-   (long unsigned int)addme[*i].bcnt);
-#endif
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t size;
+ struct compat_ipt_entry entrytable[0];
+};

- ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+static int compat_copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table, void __user *userptr)
+{
+ unsigned int off, num;
+ struct compat_ipt_entry e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ void __user *pos;
+ unsigned int size;
+ int ret = 0;
```

```
+ void *loc_cpu_entry;

- (*i)++;
- return 0;
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
+ /* choose the copy that is on our node/cpu, ...
+  * This choice is lazy (because current thread is
+  * allowed to migrate to another cpu)
+  */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ pos = userptr;
+ size = total_size;
+ ret = IPT_ENTRY_ITERATE(loc_cpu_entry, total_size,
+   compat_copy_entry_to_user, &pos, &size);
+ if (ret)
+  goto free_counters;
+
+ /* ... then go back and fix counters and names */
+ for (off = 0, num = 0; off < size; off += e.next_offset, num++) {
+  unsigned int i;
+  struct ipt_entry_match m;
+  struct ipt_entry_target t;
+
+  ret = -EFAULT;
+  if (copy_from_user(&e, userptr + off,
+     sizeof(struct compat_ipt_entry)))
+   goto free_counters;
+  if (copy_to_user(userptr + off +
+   offsetof(struct compat_ipt_entry, counters),
+    &counters[num], sizeof(counters[num])))
+   goto free_counters;
+
+  for (i = sizeof(struct compat_ipt_entry);
+    i < e.target_offset; i += m.u.match_size) {
+   if (copy_from_user(&m, userptr + off + i,
+     sizeof(struct ipt_entry_match)))
+    goto free_counters;
+   if (copy_to_user(userptr + off + i +
+    offsetof(struct ipt_entry_match, u.user.name),
+    m.u.kernel.match->name,
+    strlen(m.u.kernel.match->name) + 1))
+    goto free_counters;
+  }
+
+  if (copy_from_user(&t, userptr + off + e.target_offset,
```

```
+     sizeof(struct ipt_entry_target)))
+   goto free_counters;
+  if (copy_to_user(userptr + off + e.target_offset +
+    offsetof(struct ipt_entry_target, u.user.name),
+    t.u.kernel.target->name,
+    strlen(t.u.kernel.target->name) + 1))
+    goto free_counters;
+ }
+ ret = 0;
+free_counters:
+ vfree(counters);
+ return ret;
 }

 static int
-do_add_counters(void __user *user, unsigned int len)
+compat_get_entries(struct compat_ipt_get_entries __user *uptr, int *len)
 {
- unsigned int i;
- struct xt_counters_info tmp, *paddc;
+ int ret;
+ struct compat_ipt_get_entries get;
  struct ipt_table *t;
- struct xt_table_info *private;
- int ret = 0;
- void *loc_cpu_entry;

- if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
-  return -EFAULT;

- if (len != sizeof(tmp) + tmp.num_counters*sizeof(struct xt_counters))
+ if (*len < sizeof(get)) {
+  duprintf("compat_get_entries: %u < %u\n",
+    *len, (unsigned int)sizeof(get));
   return -EINVAL;
+ }

- paddc = vmalloc_node(len, numa_node_id());
- if (!paddc)
-  return -ENOMEM;
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;

- if (copy_from_user(paddc, user, len) != 0) {
-  ret = -EFAULT;
-  goto free;
+ if (*len != sizeof(struct compat_ipt_get_entries) + get.size) {
+  duprintf("compat_get_entries: %u != %u\n", *len,
```

```
+   (unsigned int)(sizeof(struct compat_ipt_get_entries) +
+   get.size));
+  return -EINVAL;
  }

- t = xt_find_table_lock(AF_INET, tmp.name);
- if (!t || IS_ERR(t)) {
+ xt_compat_lock(AF_INET);
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  struct xt_table_info info;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  ret = compat_table_info(private, &info);
+  if (!ret && get.size == info.size) {
+   ret = compat_copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  } else if (!ret) {
+   duprintf("compat_get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  compat_flush_offsets();
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
   ret = t ? PTR_ERR(t) : -ENOENT;
-  goto free;
- }

- write_lock_bh(&t->lock);
- private = t->private;
- if (private->number != paddc->num_counters) {
-  ret = -EINVAL;
-  goto unlock_up_free;
- }
+ xt_compat_unlock(AF_INET);
+ return ret;
+}

- i = 0;
- /* Choose the copy that is on our node */
- loc_cpu_entry = private->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_entry,
-     private->size,
-     add_counter_to_entry,
```

```
-     paddc->counters,
-     &i);
- unlock_up_free:
- write_unlock_bh(&t->lock);
- xt_table_unlock(t);
- module_put(t->me);
- free:
- vfree(paddc);
+static int
+compat_do_ipt_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
+{
+ int ret;

+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 1);
+  break;
+ case IPT_SO_GET_ENTRIES:
+  ret = compat_get_entries(user, len);
+  break;
+ default:
+  duprintf("compat_do_ipt_get_ctl: unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }
  return ret;
 }
+#endif

 static int
 do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user, unsigned int len)
@@ -1109,7 +2028,7 @@ do_ipt_set_ctl(struct sock *sk, int cmd,
   break;

  case IPT_SO_SET_ADD_COUNTERS:
-  ret = do_add_counters(user, len);
+  ret = do_add_counters(user, len, 0);
   break;

  default:
@@ -1129,65 +2048,13 @@ do_ipt_get_ctl(struct sock *sk, int cmd,
   return -EPERM;

  switch (cmd) {
- case IPT_SO_GET_INFO: {
-  char name[IPT_TABLE_MAXNAMELEN];
-  struct ipt_table *t;
-
-  if (*len != sizeof(struct ipt_getinfo)) {
```

```
-   duprintf("length %u != %u\n", *len,
-     sizeof(struct ipt_getinfo));
-   ret = -EINVAL;
-   break;
-  }
-
-  if (copy_from_user(name, user, sizeof(name)) != 0) {
-   ret = -EFAULT;
-   break;
-  }
-  name[IPT_TABLE_MAXNAMELEN-1] = '\0';
-
-  t = try_then_request_module(xt_find_table_lock(AF_INET, name),
-         "iptable_%s", name);
-  if (t && !IS_ERR(t)) {
-   struct ipt_getinfo info;
-   struct xt_table_info *private = t->private;
-
-   info.valid_hooks = t->valid_hooks;
-   memcpy(info.hook_entry, private->hook_entry,
-          sizeof(info.hook_entry));
-   memcpy(info.underflow, private->underflow,
-          sizeof(info.underflow));
-   info.num_entries = private->number;
-   info.size = private->size;
-   memcpy(info.name, name, sizeof(info.name));
-
-   if (copy_to_user(user, &info, *len) != 0)
-    ret = -EFAULT;
-   else
-    ret = 0;
-   xt_table_unlock(t);
-   module_put(t->me);
-  } else
-   ret = t ? PTR_ERR(t) : -ENOENT;
- }
- break;
-
- case IPT_SO_GET_ENTRIES: {
-  struct ipt_get_entries get;
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 0);
+  break;

-  if (*len < sizeof(get)) {
-   duprintf("get_entries: %u < %u\n", *len, sizeof(get));
-   ret = -EINVAL;
-  } else if (copy_from_user(&get, user, sizeof(get)) != 0) {
```

```
-   ret = -EFAULT;
-  } else if (*len != sizeof(struct ipt_get_entries) + get.size) {
-   duprintf("get_entries: %u != %u\n", *len,
-     sizeof(struct ipt_get_entries) + get.size);
-   ret = -EINVAL;
-  } else
-   ret = get_entries(&get, user);
+ case IPT_SO_GET_ENTRIES:
+  ret = get_entries(user, len);
    break;
- }

  case IPT_SO_GET_REVISION_MATCH:
  case IPT_SO_GET_REVISION_TARGET: {
@@ -1336,6 +2203,9 @@ static struct ipt_target ipt_standard_ta
  .name  = IPT_STANDARD_TARGET,
  .targetsize = sizeof(int),
  .family  = AF_INET,
+#ifdef CONFIG_COMPAT
+ .compat  = &compat_ipt_standard_fn,
+#endif
 };

 static struct ipt_target ipt_error_target = {
@@ -1350,9 +2220,11 @@ static struct nf_sockopt_ops ipt_sockopt
  .set_optmin = IPT_BASE_CTL,
  .set_optmax = IPT_SO_SET_MAX+1,
  .set  = do_ipt_set_ctl,
+ .compat_set = compat_do_ipt_set_ctl,
  .get_optmin = IPT_BASE_CTL,
  .get_optmax = IPT_SO_GET_MAX+1,
  .get  = do_ipt_get_ctl,
+ .compat_get = compat_do_ipt_get_ctl,
 };

 static struct ipt_match icmp_matchstruct = {
diff --git a/net/netfilter/x_tables.c b/net/netfilter/x_tables.c
index a657ab5..feb8a9e 100644
--- a/net/netfilter/x_tables.c
+++ b/net/netfilter/x_tables.c
@@ -38,6 +38,7 @@ struct xt_af {
  struct list_head match;
  struct list_head target;
  struct list_head tables;
+ struct mutex compat_mutex;
 };

 static struct xt_af *xt;
```

```
@@ -272,6 +273,54 @@ int xt_check_match(const struct xt_match
 }
 EXPORT_SYMBOL_GPL(xt_check_match);

+#ifdef CONFIG_COMPAT
+int xt_compat_match(void *match, void **dstptr, int *size, int convert)
+{
+ struct xt_match *m;
+ struct compat_xt_entry_match *pcompat_m;
+ struct xt_entry_match *pm;
+ u_int16_t msize;
+ int off, ret;
+
+ ret = 0;
+ m = ((struct xt_entry_match *)match)->u.kernel.match;
+ off = XT_ALIGN(m->matchsize) - COMPAT_XT_ALIGN(m->matchsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pm = (struct xt_entry_match *)match;
+   msize = pm->u.user.match_size;
+   if (__copy_to_user(*dstptr, pm, msize)) {
+    ret = -EFAULT;
+    break;
+   }
+   msize -= off;
+   if (put_user(msize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += msize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_m = (struct compat_xt_entry_match *)match;
+   pm = (struct xt_entry_match *)*dstptr;
+   msize = pcompat_m->u.user.match_size;
+   memcpy(pm, pcompat_m, msize);
+   msize += off;
+   pm->u.user.match_size = msize;
+   *size += off;
+   *dstptr += msize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
```

```
+}
+EXPORT_SYMBOL_GPL(xt_compat_match);
+#endif
+
 int xt_check_target(const struct xt_target *target, unsigned short family,
       unsigned int size, const char *table, unsigned int hook_mask,
       unsigned short proto, int inv_proto)
@@ -301,6 +350,54 @@ int xt_check_target(const struct xt_targ
 }
 EXPORT_SYMBOL_GPL(xt_check_target);

+#ifdef CONFIG_COMPAT
+int xt_compat_target(void *target, void **dstptr, int *size, int convert)
+{
+ struct xt_target *t;
+ struct compat_xt_entry_target *pcompat;
+ struct xt_entry_target *pt;
+ u_int16_t tsize;
+ int off, ret;
+
+ ret = 0;
+ t = ((struct xt_entry_target *)target)->u.kernel.target;
+ off = XT_ALIGN(t->targetsize) - COMPAT_XT_ALIGN(t->targetsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pt = (struct xt_entry_target *)target;
+   tsize = pt->u.user.target_size;
+   if (__copy_to_user(*dstptr, pt, tsize)) {
+    ret = -EFAULT;
+    break;
+   }
+   tsize -= off;
+   if (put_user(tsize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat = (struct compat_xt_entry_target *)target;
+   pt = (struct xt_entry_target *)*dstptr;
+   tsize = pcompat->u.user.target_size;
+   memcpy(pt, pcompat, tsize);
+   tsize += off;
+   pt->u.user.target_size = tsize;
+   *size += off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_CALC_SIZE:
```

```
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(xt_compat_target);
+#endif
+
 struct xt_table_info *xt_alloc_table_info(unsigned int size)
 {
  struct xt_table_info *newinfo;
@@ -371,6 +468,19 @@ void xt_table_unlock(struct xt_table *ta
 }
 EXPORT_SYMBOL_GPL(xt_table_unlock);

+#ifdef CONFIG_COMPAT
+void xt_compat_lock(int af)
+{
+ mutex_lock(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_lock);
+
+void xt_compat_unlock(int af)
+{
+ mutex_unlock(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_unlock);
+#endif

 struct xt_table_info *
 xt_replace_table(struct xt_table *table,
@@ -671,6 +781,9 @@ static int __init xt_init(void)

  for (i = 0; i < NPROTO; i++) {
   mutex_init(&xt[i].mutex);
+#ifdef CONFIG_COMPAT
+  mutex_init(&xt[i].compat_mutex);
+#endif
   INIT_LIST_HEAD(&xt[i].target);
   INIT_LIST_HEAD(&xt[i].match);
   INIT_LIST_HEAD(&xt[i].tables);
```

File Attachments

1) diff-ms-netfilter-ipt-compat-20060329, downloaded 533 times