Subject: [PATCH 1/2] iptables 32bit compat layer
Posted by Mishin Dmitry on Mon, 20 Feb 2006 08:10:38 GMT
View Forum Message <> Reply to Message

Hello,

This patch set 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.

This patch introduces base compatibility interface for other ip_tables modules

--
Thanks,
Dmitry.

```
--- ./include/linux/netfilter/x_tables.h.iptcompat 2006-02-15 16:16:02.000000000 +0300
+++ ./include/linux/netfilter/x_tables.h 2006-02-15 18:53:09.000000000 +0300
@@ -80,12 +80,19 @@ struct xt_counters_info

 #ifdef __KERNEL__

+#include <linux/config.h>
 #include <linux/netdevice.h>

 #define ASSERT_READ_LOCK(x)
 #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;
@@ -118,6 +125,10 @@ struct xt_match
  /* Called when entry of this type deleted. */
  void (*destroy)(void *matchinfo, unsigned int matchinfosize);

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

```
 };
@@ -154,6 +165,10 @@ struct xt_target
  /* Called when entry of this type deleted. */
  void (*destroy)(void *targinfo, unsigned int targinfosize);

+#ifdef CONFIG_COMPAT
+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *target, void **dstptr, int *size, int convert);
+#endif
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;
 };
@@ -233,6 +248,34 @@ 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>
+
+/* 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 int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert);
+extern int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /* __KERNEL__ */

 #endif /* _X_TABLES_H */
--- ./include/linux/netfilter_ipv4/ip_tables.h.iptcompat 2006-02-15 16:06:41.000000000 +0300
```

```
+++ ./include/linux/netfilter_ipv4/ip_tables.h 2006-02-15 16:37:12.000000000 +0300
@@ -16,6 +16,7 @@
 #define _IPTABLES_H

 #ifdef __KERNEL__
+#include <linux/config.h>
 #include <linux/if.h>
 #include <linux/types.h>
 #include <linux/in.h>
@@ -364,5 +365,62 @@ 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_getinfo
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t valid_hooks;
+ compat_uint_t hook_entry[NF_IP_NUMHOOKS];
+ compat_uint_t underflow[NF_IP_NUMHOOKS];
+ compat_uint_t num_entries;
+ compat_uint_t size;
+};
+
+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];
+};
+
+struct compat_ipt_entry_match
+{
+ union {
+  struct {
+   u_int16_t match_size;
+   char name[IPT_FUNCTION_MAXNAMELEN];
+  } user;
+  u_int16_t match_size;
+ } u;
+ unsigned char data[0];
```

```
+};
+
+struct compat_ipt_entry_target
+{
+ union {
+  struct {
+   u_int16_t target_size;
+   char name[IPT_FUNCTION_MAXNAMELEN];
+  } user;
+  u_int16_t target_size;
+ } u;
+ unsigned char data[0];
+};
+
+#define COMPAT_IPT_ALIGN(s)  COMPAT_XT_ALIGN(s)
+
+extern int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert);
+extern int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /*__KERNEL__*/
 #endif /* _IPTABLES_H */
--- ./include/net/compat.h.iptcompat 2006-01-03 06:21:10.000000000 +0300
+++ ./include/net/compat.h 2006-02-15 18:45:49.000000000 +0300
@@ -23,6 +23,14 @@ struct compat_cmsghdr {
  compat_int_t cmsg_type;
 };

+#if defined(CONFIG_X86_64)
+#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)
+#elif defined(CONFIG_IA64)
+#define is_current_32bits() (IS_IA32_PROCESS(ia64_task_regs(current)))
+#else
+#define is_current_32bits() 0
+#endif
+
 #else /* defined(CONFIG_COMPAT) */
 #define compat_msghdr msghdr  /* to avoid compiler warnings */
 #endif /* defined(CONFIG_COMPAT) */
--- ./net/compat.c.iptcompat 2006-01-03 06:21:10.000000000 +0300
+++ ./net/compat.c 2006-02-15 16:38:45.000000000 +0300
@@ -308,107 +308,6 @@ void scm_detach_fds_compat(struct msghdr
 }

 /*
- * For now, we assume that the compatibility and native version
```

```
- * of struct ipt_entry are the same - sfr.  FIXME
- */
-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 ipt_entry entries[0];
-};
-
-static int do_netfilter_replace(int fd, int level, int optname,
-    char __user *optval, int optlen)
-{
- struct compat_ipt_replace __user *urepl;
- struct ipt_replace __user *repl_nat;
- char name[IPT_TABLE_MAXNAMELEN];
- u32 origsize, tmp32, num_counters;
- unsigned int repl_nat_size;
- int ret;
- int i;
- compat_uptr_t ucntrs;
-
- urepl = (struct compat_ipt_replace __user *)optval;
- if (get_user(origsize, &urepl->size))
-  return -EFAULT;
-
- /* Hack: Causes ipchains to give correct error msg --RR */
- if (optlen != sizeof(*urepl) + origsize)
-  return -ENOPROTOOPT;
-
- /* XXX Assumes that size of ipt_entry is the same both in
-  *     native and compat environments.
-  */
- repl_nat_size = sizeof(*repl_nat) + origsize;
- repl_nat = compat_alloc_user_space(repl_nat_size);
-
- ret = -EFAULT;
- if (put_user(origsize, &repl_nat->size))
-  goto out;
-
- if (!access_ok(VERIFY_READ, urepl, optlen) ||
-     !access_ok(VERIFY_WRITE, repl_nat, optlen))
-  goto out;
-
```

```
- if (__copy_from_user(name, urepl->name, sizeof(urepl->name)) ||
-     __copy_to_user(repl_nat->name, name, sizeof(repl_nat->name)))
-  goto out;
-
- if (__get_user(tmp32, &urepl->valid_hooks) ||
-     __put_user(tmp32, &repl_nat->valid_hooks))
-  goto out;
-
- if (__get_user(tmp32, &urepl->num_entries) ||
-     __put_user(tmp32, &repl_nat->num_entries))
-  goto out;
-
- if (__get_user(num_counters, &urepl->num_counters) ||
-     __put_user(num_counters, &repl_nat->num_counters))
-  goto out;
-
- if (__get_user(ucntrs, &urepl->counters) ||
-     __put_user(compat_ptr(ucntrs), &repl_nat->counters))
-  goto out;
-
- if (__copy_in_user(&repl_nat->entries[0],
-      &urepl->entries[0],
-      origsize))
-  goto out;
-
- for (i = 0; i < NF_IP_NUMHOOKS; i++) {
-  if (__get_user(tmp32, &urepl->hook_entry[i]) ||
-      __put_user(tmp32, &repl_nat->hook_entry[i]) ||
-      __get_user(tmp32, &urepl->underflow[i]) ||
-      __put_user(tmp32, &repl_nat->underflow[i]))
-   goto out;
- }
-
- /*
-  * Since struct ipt_counters just contains two u_int64_t members
-  * we can just do the access_ok check here and pass the (converted)
-  * pointer into the standard syscall.  We hope that the pointer is
-  * not misaligned ...
-  */
- if (!access_ok(VERIFY_WRITE, compat_ptr(ucntrs),
-         num_counters * sizeof(struct ipt_counters)))
-  goto out;
-
-
- ret = sys_setsockopt(fd, level, optname,
-        (char __user *)repl_nat, repl_nat_size);
-
-out:
```

```
- return ret;
-}
-
-/*
  * A struct sock_filter is architecture independent.
  */
 struct compat_sock_fprog {
@@ -460,10 +359,6 @@ static int do_set_sock_timeout(int fd, i
 asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
     char __user *optval, int optlen)
 {
- /* SO_SET_REPLACE seems to be the same in all levels */
- if (optname == IPT_SO_SET_REPLACE)
-  return do_netfilter_replace(fd, level, optname,
-         optval, optlen);
  if (level == SOL_SOCKET && optname == SO_ATTACH_FILTER)
   return do_set_attach_filter(fd, level, optname,
          optval, optlen);
--- ./net/ipv4/netfilter/ip_tables.c.iptcompat 2006-02-15 16:06:42.000000000 +0300
+++ ./net/ipv4/netfilter/ip_tables.c 2006-02-17 19:38:05.000000000 +0300
@@ -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 <asm/semaphore.h>
 #include <linux/proc_fs.h>
@@ -480,7 +481,7 @@ standard_check(const struct ipt_entry_ta
  if (t->u.target_size
      != IPT_ALIGN(sizeof(struct ipt_standard_target))) {
   duprintf("standard_check: target size %u != %u\n",
-    t->u.target_size,
+    t->u.target_size, (unsigned int)
     IPT_ALIGN(sizeof(struct ipt_standard_target)));
   return 0;
  }
@@ -790,17 +791,11 @@ 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
@@ -809,13 +804,32 @@ 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)
@@ -875,25 +889,391 @@ copy_entries_to_user(unsigned int total_
  return ret;
 }

+#ifdef CONFIG_COMPAT
```

```
+static DECLARE_MUTEX(compat_ipt_mutex);
+
+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_ipt_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;
+}
+
+int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert)
+{
+ struct compat_ipt_entry_target *pcompat;
+ struct ipt_entry_target *pt;
+ u_int16_t tsize;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pt = (struct ipt_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_ipt_entry_target *)target;
+   pt = (struct ipt_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;
+}
+
+int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert)
+{
+ struct compat_ipt_entry_match *pcompat_m;
+ struct ipt_entry_match *pm;
+ u_int16_t msize;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pm = (struct ipt_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_ipt_entry_match *)match;
+   pm = (struct ipt_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;
+}
+
+static int icmp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = IPT_ALIGN(sizeof(struct ipt_icmp)) -
+  COMPAT_IPT_ALIGN(sizeof(struct ipt_icmp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+
+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);
+ 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);
+ 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;
+ 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)
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ struct ipt_table *t;
+ int ret, size;
+
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits())
+  size = sizeof(struct compat_ipt_getinfo);
+ else
+#endif
+  size = sizeof(struct ipt_getinfo);
+
+ if (*len != size) {
+  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
+ down(&compat_ipt_mutex);
+#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
+  struct compat_ipt_getinfo compat_info;
+#endif
+  void *pinfo;
+
+#ifdef CONFIG_COMPAT
+  if (is_current_32bits()) {
+   struct xt_table_info tmp;
+   ret = compat_table_info(private, &tmp);
+   compat_flush_offsets();
+   memcpy(compat_info.hook_entry, tmp.hook_entry,
+     sizeof(compat_info.hook_entry));
+   memcpy(compat_info.underflow, tmp.underflow,
+     sizeof(compat_info.underflow));
+   compat_info.valid_hooks = t->valid_hooks;
+   compat_info.num_entries = private->number;
+   compat_info.size = tmp.size;
+   strcpy(compat_info.name, name);
+   pinfo = (void *)&compat_info;
+  } else
+#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);
+   pinfo = (void *)&info;
+  }
+
```

```
+  if (copy_to_user(user, pinfo, *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
+  up(&compat_ipt_mutex);
+#endif
+ return ret;
+}
+
 static int
-get_entries(const struct ipt_get_entries *entries,
-     struct ipt_get_entries __user *uptr)
+get_entries(struct ipt_get_entries __user *uptr, int *len)
 {
  int ret;
+ struct ipt_get_entries get;
  struct ipt_table *t;

- t = xt_find_table_lock(AF_INET, entries->name);
+ 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 (entries->size == private->size)
+  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,
-     entries->size);
+     get.size);
    ret = -EINVAL;
   }
   module_put(t->me);
@@ -905,71 +1285,39 @@ get_entries(const struct ipt_get_entries
 }

 static int
-do_replace(void __user *user, unsigned int len)
+__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_replace tmp;
  struct ipt_table *t;
- struct xt_table_info *newinfo, *oldinfo;
+ struct xt_table_info *oldinfo;
  struct xt_counters *counters;
- void *loc_cpu_entry, *loc_cpu_old_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;
- }
+ void *loc_cpu_old_entry;

- counters = vmalloc(tmp.num_counters * sizeof(struct xt_counters));
+ ret = 0;
+ counters = vmalloc(num_counters * sizeof(struct xt_counters));
  if (!counters) {
   ret = -ENOMEM;
-  goto free_newinfo;
+  goto out;
  }

- 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("ip_tables: Translated table\n");
-
- t = try_then_request_module(xt_find_table_lock(AF_INET, tmp.name),
-         "iptable_%s", tmp.name);
+ 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 (tmp.valid_hooks != t->valid_hooks) {
+ if (valid_hooks != t->valid_hooks) {
   duprintf("Valid hook crap: %08X vs %08X\n",
-    tmp.valid_hooks, t->valid_hooks);
+    valid_hooks, t->valid_hooks);
   ret = -EINVAL;
   goto put_module;
  }

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

@@ -989,8 +1337,8 @@ do_replace(void __user *user, unsigned i
  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)
+ if (copy_to_user(counters_ptr, counters,
+     sizeof(struct xt_counters) * num_counters) != 0)
    ret = -EFAULT;
  vfree(counters);
  xt_table_unlock(t);
@@ -1000,9 +1348,62 @@ do_replace(void __user *user, unsigned i
  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);
+ 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;
@@ -1034,28 +1435,56 @@ static int
 do_add_counters(void __user *user, unsigned int len)
 {
  unsigned int i;
- struct xt_counters_info tmp, *paddc;
+ 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 (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+ if (is_current_32bits()) {
+  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;

- if (len != sizeof(tmp) + tmp.num_counters*sizeof(struct xt_counters))
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits()) {
+  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, numa_node_id());
+ paddc = vmalloc_node(len - size, numa_node_id());
  if (!paddc)
   return -ENOMEM;

- if (copy_from_user(paddc, user, len) != 0) {
+ if (copy_from_user(paddc, user + size, len - size) != 0) {
   ret = -EFAULT;
   goto free;
  }

- t = xt_find_table_lock(AF_INET, tmp.name);
+ t = xt_find_table_lock(AF_INET, name);
  if (!t || IS_ERR(t)) {
   ret = t ? PTR_ERR(t) : -ENOENT;
   goto free;
@@ -1063,7 +1492,7 @@ do_add_counters(void __user *user, unsig

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

```
 }
@@ -1074,7 +1503,7 @@ do_add_counters(void __user *user, unsig
  IPT_ENTRY_ITERATE(loc_cpu_entry,
      private->size,
      add_counter_to_entry,
-     paddc->counters,
+     paddc,
      &i);
  unlock_up_free:
  write_unlock_bh(&t->lock);
@@ -1086,6 +1515,577 @@ do_add_counters(void __user *user, unsig
  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)
+  m->u.kernel.match->compat(m, dstptr, size, COMPAT_TO_USER);
+ else {
+  if (__copy_to_user(*dstptr, m, m->u.match_size))
+   return -EFAULT;
+  *dstptr += m->u.match_size;
+ }
+ return 0;
+}
+
+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);
+  if (ret)
+   goto out;
+ } else {
+  ret = -EFAULT;
+  if (__copy_to_user(*dstptr, t, t->u.target_size))
+   goto out;
+  *dstptr += t->u.target_size;
+ }
+ 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);
+
+ (*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_ipt_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);
+ *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;
+
+ dm = (struct ipt_entry_match *)*dstptr;
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, dstptr, size, COMPAT_FROM_USER);
+ else {
+  memcpy(*dstptr, m, m->u.match_size);
+  *dstptr += m->u.match_size;
+ }
+
+ if (dm->u.kernel.match->checkentry
+     && !dm->u.kernel.match->checkentry(name, ip, dm->data,
+           dm->u.match_size - sizeof(*dm),
+           hookmask)) {
+  module_put(dm->u.kernel.match->me);
+  duprintf("ip_tables: check failed for `%s'.\n",
+    dm->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_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);
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t,
```

```
+    dstptr, size, COMPAT_FROM_USER);
+ else {
+  memcpy(*dstptr, t, t->u.target_size);
+  *dstptr += t->u.target_size;
+ }
+
+ 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;
+ }
+
+ ret = -EINVAL;
+ t = ipt_get_target(de);
+ 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, t->data,
+            t->u.target_size
+            - sizeof(*t),
+            de->comefrom)) {
+  module_put(t->u.kernel.target->me);
+  duprintf("ip_tables: compat: check failed for `%s'.\n",
+    t->u.kernel.target->name);
+  goto out;
+ }
+ ret = 0;
+out:
+ return ret;
+}
+
+static int
+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;
+
+ 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;
+ down(&compat_ipt_mutex);
+ /* 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;
+  }
+  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();
+ up(&compat_ipt_mutex);
+ 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:
+ up(&compat_ipt_mutex);
+ goto out;
+}
+
+static int
+compat_do_replace(void __user *user, unsigned int len)
+{
```

```
+ int ret;
+ 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;
+
+ /* 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_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;
+
+ duprintf("compat_do_replace: 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;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
```

```
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}
+
+struct compat_ipt_get_entries
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t size;
+ struct compat_ipt_entry entrytable[0];
+};
+
+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;
+
+ 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
+compat_get_entries(struct compat_ipt_get_entries __user *uptr, int *len)
+{
+ int ret;
+ struct compat_ipt_get_entries get;
+ struct ipt_table *t;
+
+
+ if (*len < sizeof(get)) {
+  duprintf("compat_get_entries: %u < %u\n",
+    *len, (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+
```

```
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+
+ 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;
+ }
+
+ down(&compat_ipt_mutex);
+ 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;
+
+ up(&compat_ipt_mutex);
+ return ret;
+}
+
+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);
+  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)
 {
@@ -1094,6 +2094,11 @@ do_ipt_set_ctl(struct sock *sk, int cmd,
  if (!capable(CAP_NET_ADMIN))
   return -EPERM;

+#ifdef CONFIG_COMPAT
+ if (is_current_32bits() && (cmd == IPT_SO_SET_REPLACE))
+  return compat_do_replace(user, len);
+#endif
+
  switch (cmd) {
  case IPT_SO_SET_REPLACE:
   ret = do_replace(user, len);
@@ -1119,66 +2124,19 @@ do_ipt_get_ctl(struct sock *sk, int cmd,
  if (!capable(CAP_NET_ADMIN))
   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;
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits())
+  return compat_do_ipt_get_ctl(sk, cmd, user, len);
+#endif

- case IPT_SO_GET_ENTRIES: {
-  struct ipt_get_entries get;
+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len);
+  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: {
@@ -1327,6 +2285,9 @@ icmp_checkentry(const char *tablename,
 /* The built-in targets: standard (NULL) and error. */
 static struct ipt_target ipt_standard_target = {
  .name  = IPT_STANDARD_TARGET,
+#ifdef CONFIG_COMPAT
+ .compat  = &compat_ipt_standard_fn,
+#endif
 };

 static struct ipt_target ipt_error_target = {
@@ -1348,6 +2309,9 @@ static struct ipt_match icmp_matchstruct
  .name  = "icmp",
  .match  = &icmp_match,
  .checkentry = &icmp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &icmp_compat,
+#endif
 };

 static int __init init(void)
@@ -1386,5 +2350,9 @@ static void __exit fini(void)
 EXPORT_SYMBOL(ipt_register_table);
 EXPORT_SYMBOL(ipt_unregister_table);
 EXPORT_SYMBOL(ipt_do_table);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(ipt_match_align_compat);
+EXPORT_SYMBOL(ipt_target_align_compat);
+#endif
 module_init(init);
 module_exit(fini);
--- ./net/netfilter/xt_tcpudp.c.iptcompat 2006-02-15 16:06:42.000000000 +0300
+++ ./net/netfilter/xt_tcpudp.c 2006-02-17 19:41:58.000000000 +0300
@@ -266,10 +266,35 @@ udp6_checkentry(const char *tablename,
  return 1;
 }

+#ifdef CONFIG_COMPAT
+static int tcp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = XT_ALIGN(sizeof(struct xt_tcp)) -
```

```
+  COMPAT_XT_ALIGN(sizeof(struct xt_tcp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+
+static int udp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = XT_ALIGN(sizeof(struct xt_udp)) -
+  COMPAT_XT_ALIGN(sizeof(struct xt_udp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+#endif
+
 static struct xt_match tcp_matchstruct = {
  .name  = "tcp",
  .match  = &tcp_match,
 .checkentry = &tcp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &tcp_compat,
+#endif
 .me  = THIS_MODULE,
 };
 static struct xt_match tcp6_matchstruct = {
@@ -283,6 +308,9 @@ static struct xt_match udp_matchstruct =
 .name  = "udp",
 .match  = &udp_match,
 .checkentry = &udp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &udp_compat,
+#endif
 .me  = THIS_MODULE,
 };
 static struct xt_match udp6_matchstruct = {
```

File Attachments

1) diff-ms-ipt-compat-20060217, downloaded 870 times