Subject: [PATCH 5/8] Move some core sock setup into sk_prot_alloc
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:49:59 GMT
View Forum Message <> Reply to Message

The security_sk_alloc() and the module_get is a part of the
object allocations - move it in the proper place.

Note, that since we do not reset the newly allocated sock
in the sk_alloc() (memset() is removed with the previous
patch) we can safely do this.

Also fix the error path in sk_prot_alloc() - release the security
context if needed.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index 21fc79b..e7537e4 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -870,7 +870,8 @@ static void sock_copy(struct sock *nsk, const struct sock *osk)
 #endif
 }

-static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority)
+static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority,
+  int family)
 {
  struct sock *sk;
  struct kmem_cache *slab;
@@ -881,18 +882,40 @@ static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority)
  else
   sk = kmalloc(prot->obj_size, priority);

+ if (sk != NULL) {
+  if (security_sk_alloc(sk, family, priority))
+   goto out_free;
+
+  if (!try_module_get(prot->owner))
+   goto out_free_sec;
+ }
+
  return sk;
+
+out_free_sec:
+ security_sk_free(sk);
```

```
+out_free:
+ if (slab != NULL)
+  kmem_cache_free(slab, sk);
+ else
+  kfree(sk);
+ return NULL;
 }

 static void sk_prot_free(struct proto *prot, struct sock *sk)
 {
  struct kmem_cache *slab;
-
+ struct module *owner;
+
+ owner = prot->owner;
  slab = prot->slab;
+
+ security_sk_free(sk);
  if (slab != NULL)
   kmem_cache_free(slab, sk);
  else
   kfree(sk);
+ module_put(owner);
 }

 /**
@@ -911,7 +934,7 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
  if (zero_it)
   priority |= __GFP_ZERO;

- sk = sk_prot_alloc(prot, priority);
+ sk = sk_prot_alloc(prot, priority, family);
  if (sk) {
   if (zero_it) {
    sk->sk_family = family;
@@ -923,24 +946,14 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
    sock_lock_init(sk);
    sk->sk_net = get_net(net);
   }
-
-  if (security_sk_alloc(sk, family, priority))
-   goto out_free;
-
-  if (!try_module_get(prot->owner))
-   goto out_free;
  }
- return sk;
```

```
-out_free:
- sk_prot_free(prot, sk);
- return NULL;
+ return sk;
 }

 void sk_free(struct sock *sk)
 {
  struct sk_filter *filter;
- struct module *owner = sk->sk_prot_creator->owner;

  if (sk->sk_destruct)
   sk->sk_destruct(sk);
@@ -957,10 +970,8 @@ void sk_free(struct sock *sk)
   printk(KERN_DEBUG "%s: optmem leakage (%d bytes) detected.\n",
          __FUNCTION__, atomic_read(&sk->sk_omem_alloc));

- security_sk_free(sk);
  put_net(sk->sk_net);
  sk_prot_free(sk->sk_prot_creator, sk);
- module_put(owner);
 }

 struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
--
1.5.3.4
```