Subject: [PATCH 1/2] c/r: Do not crash if socket has no peercred
Posted by Dan Smith on Mon, 07 Feb 2011 16:42:58 GMT
View Forum Message <> Reply to Message

Unconnected sockets don't have a valid sk_peercred yet. If we checkpoint
them, we'll segv on the NULL pointer.

Signed-off-by: Dan Smith <danms@us.ibm.com>
---
net/unix/checkpoint.c | 52 ++++++++++++++++++++++++++----------------
1 files changed, 29 insertions(+), 23 deletions(-)

```
diff --git a/net/unix/checkpoint.c b/net/unix/checkpoint.c
index 708df40..25464cc 100644
--- a/net/unix/checkpoint.c
+++ b/net/unix/checkpoint.c
@@ -159,16 +159,19 @@ int unix_checkpoint(struct ckpt_ctx *ctx, struct socket *sock)
   goto out;
  }

- /*
-  * intentionally drop 'const' qualifier for checkpoint_obj() to
-  * increment the usage count - it does not alter the credentials.
-  */
- cred = (struct cred *) get_cred(sock->sk->sk_peer_cred);
- un->peercred = checkpoint_obj(ctx, cred, CKPT_OBJ_CRED);
- put_cred(cred);
- if (un->peercred < 0) {
-  ret = un->peercred;
-  goto out;
+ if (sock->sk->sk_peer_cred) {
+  /*
+   * intentionally drop 'const' qualifier for
+   * checkpoint_obj() to increment the usage count - it
+   * does not alter the credentials.
+   */
+  cred = (struct cred *) get_cred(sock->sk->sk_peer_cred);
+  un->peercred = checkpoint_obj(ctx, cred, CKPT_OBJ_CRED);
+  put_cred(cred);
+  if (un->peercred < 0) {
+   ret = un->peercred;
+   goto out;
+  }
  }

  ret = ckpt_write_obj(ctx, (struct ckpt_hdr *) un);
@@ -438,19 +441,22 @@ static int unix_restore_connected(struct ckpt_ctx *ctx,
   addrlen = un->laddr_len;
```

```
  }

- cred = ckpt_obj_fetch(ctx, un->peercred, CKPT_OBJ_CRED);
- if (!cred) {
-  ckpt_err(ctx, -EINVAL, "%(O)Bad peer cred\n", un->peercred);
-  return -EINVAL;
- }
-
- if (may_setuid(ctx->realcred->user->user_ns, cred->uid) &&
-     may_setgid(cred->gid)) {
-  set_peercred(sk, task_tgid(current), cred);
- } else {
-  ckpt_err(ctx, -EPERM, "peercred %i:%i would require setuid",
-    cred->uid, cred->gid);
-  return -EPERM;
+ if (un->peercred) {
+  cred = ckpt_obj_fetch(ctx, un->peercred, CKPT_OBJ_CRED);
+  if (!cred) {
+   ckpt_err(ctx, -EINVAL,
+     "%(O)Bad peer cred\n", un->peercred);
+   return -EINVAL;
+  }
+  if (may_setuid(ctx->realcred->user->user_ns, cred->uid) &&
+      may_setgid(cred->gid)) {
+   set_peercred(sk, task_tgid(current), cred);
+  } else {
+   ckpt_err(ctx, -EPERM,
+     "peercred %i:%i would require setuid",
+     cred->uid, cred->gid);
+   return -EPERM;
+  }
  }

  if (!dead && (un->peer > 0)) {
--
1.7.2.2
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

---

Subject: [PATCH 2/2] c/r: Check for bound AF_UNIX sockets before doing unlink()
Posted by Dan Smith on Mon, 07 Feb 2011 16:42:59 GMT
View Forum Message <> Reply to Message

If we're not restarting inside our own network namespace, the explicit

unlink() of the AF_UNIX socket path before binding is not safe. This
could end up hosing something that has that socket open. The easiest way to
encounter such a problem is to checkpoint and restart an application with a
connection to syslogd. Without this patch, on restart you hose the system
syslogd by unlink()'ing its /dev/log.

We hold the mutex on the parent inode to make sure that we don't race with
another unix_bind() in progress.

Signed-off-by: Dan Smith <danms@us.ibm.com>
---
 include/net/af_unix.h | 1 +
 net/unix/af_unix.c | 2 +-
 net/unix/checkpoint.c | 55 ++++++++++++++++++++++++++----------------------
 3 files changed, 28 insertions(+), 30 deletions(-)

diff --git a/include/net/af_unix.h b/include/net/af_unix.h
index add734f..a24a842 100644
--- a/include/net/af_unix.h
+++ b/include/net/af_unix.h
@@ -11,6 +11,7 @@ extern void unix_notinflight(struct file *fp);
 extern void unix_gc(void);
 extern void wait_for_unix_gc(void);
 extern struct sock *unix_get_socket(struct file *filp);
+extern struct sock *unix_find_socket_byinode(struct inode *i);

 #define UNIX_HASH_SIZE 256

diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index cb7afd5..e385a1e 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -282,7 +282,7 @@ static inline struct sock *unix_find_socket_byname(struct net *net,
  return s;
 }

-static struct sock *unix_find_socket_byinode(struct inode *i)
+struct sock *unix_find_socket_byinode(struct inode *i)
 {
  struct sock *s;
  struct hlist_node *node;
diff --git a/net/unix/checkpoint.c b/net/unix/checkpoint.c
index 25464cc..0a06c94 100644
--- a/net/unix/checkpoint.c
+++ b/net/unix/checkpoint.c
@@ -467,38 +467,32 @@ static int unix_restore_connected(struct ckpt_ctx *ctx,
  return ret;
 }

```
-static int unix_unlink(const char *name)
+/**
+ * Perform the unlink() operation of a hopefully-stale unix socket.
+ * We should, however, first determine that a socket is not already
+ * bound there.  If not, we can unlink() while holding the parent's
+ * inode mutex to make sure that we don't race with bind().
+ */
+static int unix_unlink(struct path *parent, const char *name)
 {
- struct path spath;
- struct path ppath;
- int ret;
+ int ret = -EADDRINUSE;
+ struct sock *sk;
+ struct path path;

- ret = kern_path(name, 0, &spath);
+ ret = kern_path(name, LOOKUP_OPEN, &path);
  if (ret)
-  return ret;
+  return 0; /* Not found is a good thing! */

- ret = kern_path(name, LOOKUP_PARENT, &ppath);
- if (ret)
-  goto out_s;
+ mutex_lock(&parent->dentry->d_inode->i_mutex);

- if (!spath.dentry) {
-  ckpt_debug("No dentry found for %s\n", name);
-  ret = -ENOENT;
-  goto out_p;
- }
-
- if (!ppath.dentry || !ppath.dentry->d_inode) {
-  ckpt_debug("No inode for parent of %s\n", name);
-  ret = -ENOENT;
-  goto out_p;
- }
-
- ret = vfs_unlink(ppath.dentry->d_inode, spath.dentry);
- out_p:
- path_put(&ppath);
- out_s:
- path_put(&spath);
+ sk = unix_find_socket_byinode(path.dentry->d_inode);
+ if (sk)
+  sock_put(sk);
```

```
+ else
+  ret = vfs_unlink(parent->dentry->d_inode, path.dentry);

+ mutex_unlock(&parent->dentry->d_inode->i_mutex);
+ path_put(&path);
  return ret;
 }

@@ -531,9 +525,13 @@ static int unix_chdir_and_bind(struct socket *sock,
   cur = current->fs->pwd;
   current->fs->pwd = dir;
   spin_unlock(&current->fs->lock);
+ } else {
+  ret = kern_path(un->sun_path, LOOKUP_PARENT, &dir);
+  if (ret)
+   return ret;
  }

- ret = unix_unlink(un->sun_path);
+ ret = unix_unlink(&dir, un->sun_path);
  ckpt_debug("unlink(%s): %i\n", un->sun_path, ret);
  if ((ret == 0) || (ret == -ENOENT))
   ret = sock_bind(sock, addr, addrlen);
@@ -544,8 +542,7 @@ static int unix_chdir_and_bind(struct socket *sock,
   spin_unlock(&current->fs->lock);
  }
  out:
- if (path)
-  path_put(&dir);
+ path_put(&dir);

  return ret;
 }
--
1.7.2.2
```