Subject: [PATCH] eCryptfs: Remove obsolete netlink interface to daemon
Posted by Michael Halcrow on Wed, 16 Apr 2008 21:10:56 GMT
View Forum Message <> Reply to Message

Remove the obsolete and buggy netlink interface to the userspace
daemon.

Signed-off-by: Michael Halcrow <mhalcrow@us.ibm.com>
---
 fs/ecryptfs/Makefile | 2 +-
 fs/ecryptfs/ecryptfs_kernel.h | 12 --
 fs/ecryptfs/main.c | 15 +--
 fs/ecryptfs/messaging.c | 31 ++----
 fs/ecryptfs/netlink.c | 249 --
 5 files changed, 17 insertions(+), 292 deletions(-)
 delete mode 100644 fs/ecryptfs/netlink.c

diff --git a/fs/ecryptfs/Makefile b/fs/ecryptfs/Makefile
index 1e34a7f..aa22276 100644
--- a/fs/ecryptfs/Makefile
+++ b/fs/ecryptfs/Makefile
@@ -4,4 +4,4 @@

 obj-$(CONFIG_ECRYPT_FS) += ecryptfs.o

-ecryptfs-objs := dentry.o file.o inode.o main.o super.o mmap.o read_write.o crypto.o keystore.o
messaging.o netlink.o miscdev.o debug.o
+ecryptfs-objs := dentry.o file.o inode.o main.o super.o mmap.o read_write.o crypto.o keystore.o
messaging.o miscdev.o debug.o
diff --git a/fs/ecryptfs/ecryptfs_kernel.h b/fs/ecryptfs/ecryptfs_kernel.h
index dc11431..661f202 100644
--- a/fs/ecryptfs/ecryptfs_kernel.h
+++ b/fs/ecryptfs/ecryptfs_kernel.h
@@ -625,18 +625,6 @@ int ecryptfs_wait_for_response(struct ecryptfs_msg_ctx *msg_ctx,
          struct ecryptfs_message **emsg);
 int ecryptfs_init_messaging(unsigned int transport);
 void ecryptfs_release_messaging(unsigned int transport);
-
-int ecryptfs_send_netlink(char *data, int data_len,
-     struct ecryptfs_msg_ctx *msg_ctx, u8 msg_type,
-     u16 msg_flags, struct pid *daemon_pid);
-int ecryptfs_init_netlink(void);
-void ecryptfs_release_netlink(void);
-
-int ecryptfs_send_connector(char *data, int data_len,
-       struct ecryptfs_msg_ctx *msg_ctx, u8 msg_type,
-       u16 msg_flags, struct pid *daemon_pid);
-int ecryptfs_init_connector(void);

```
-void ecryptfs_release_connector(void);
 void
 ecryptfs_write_header_metadata(char *virt,
          struct ecryptfs_crypt_stat *crypt_stat,
diff --git a/fs/ecryptfs/main.c b/fs/ecryptfs/main.c
index d25ac95..55fee42 100644
--- a/fs/ecryptfs/main.c
+++ b/fs/ecryptfs/main.c
@@ -30,7 +30,6 @@
 #include <linux/namei.h>
 #include <linux/skbuff.h>
 #include <linux/crypto.h>
-#include <linux/netlink.h>
 #include <linux/mount.h>
 #include <linux/pagemap.h>
 #include <linux/key.h>
@@ -49,8 +48,7 @@ MODULE_PARM_DESC(ecryptfs_verbosity,
    "0, which is Quiet)");

 /**
- * Module parameter that defines the number of netlink message buffer
- * elements
+ * Module parameter that defines the number of message buffer elements
  */
 unsigned int ecryptfs_message_buf_len = ECRYPTFS_DEFAULT_MSG_CTX_ELEMS;

@@ -59,10 +57,8 @@ MODULE_PARM_DESC(ecryptfs_message_buf_len,
    "Number of message buffer elements");

 /**
- * Module parameter that defines the maximum guaranteed amount of time to wait
- * for a response through netlink.  The actual sleep time will be, more than
- * likely, a small amount greater than this specified value, but only less if
- * the netlink message successfully arrives.
+ * Module parameter that defines the maximum guaranteed amount of time
+ * to wait for a response from the userspace daemon.
  */
 signed long ecryptfs_message_wait_timeout = ECRYPTFS_MAX_MSG_CTX_TTL / HZ;

@@ -797,8 +793,9 @@ static int __init ecryptfs_init(void)
  }
  rc = ecryptfs_init_messaging(ecryptfs_transport);
  if (rc) {
-  ecryptfs_printk(KERN_ERR, "Failure occured while attempting to "
-    "initialize the eCryptfs netlink socket\n");
+  printk(KERN_ERR "%s: Failure occured while attempting to "
+         "initialize the eCryptfs daemon messaging subsystem; "
+         "rc = [%d]\n", __func__, rc);
```

```
   goto out_do_sysfs_unregistration;
  }
  rc = ecryptfs_init_crypto();
diff --git a/fs/ecryptfs/messaging.c b/fs/ecryptfs/messaging.c
index fad161b..be1622d 100644
--- a/fs/ecryptfs/messaging.c
+++ b/fs/ecryptfs/messaging.c
@@ -155,16 +155,13 @@ static int ecryptfs_send_raw_message(unsigned int transport, u8
msg_type,
  int rc;

  switch(transport) {
- case ECRYPTFS_TRANSPORT_NETLINK:
-  rc = ecryptfs_send_netlink(NULL, 0, NULL, msg_type, 0,
-         daemon->pid);
-  break;
  case ECRYPTFS_TRANSPORT_MISCDEV:
   rc = ecryptfs_send_message_locked(transport, NULL, 0, msg_type,
         &msg_ctx);
   if (rc) {
    printk(KERN_ERR "%s: Error whilst attempting to send "
-          "message via procfs; rc = [%d]\n", __func__, rc);
+          "message via device handle; rc = [%d]\n",
+          __func__, rc);
    goto out;
   }
   /* Raw messages are logically context-free (e.g., no
@@ -175,6 +172,7 @@ static int ecryptfs_send_raw_message(unsigned int transport, u8
msg_type,
   msg_ctx->state = ECRYPTFS_MSG_CTX_STATE_NO_REPLY;
   mutex_unlock(&msg_ctx->mux);
   break;
+ case ECRYPTFS_TRANSPORT_NETLINK:
  case ECRYPTFS_TRANSPORT_CONNECTOR:
  case ECRYPTFS_TRANSPORT_RELAYFS:
  default:
@@ -225,7 +223,7 @@ out:

 /**
  * ecryptfs_process_helo
- * @transport: The underlying transport (netlink, etc.)
+ * @transport: The underlying transport (miscdev, etc.)
  * @euid: The user ID owner of the message
  * @user_ns: The namespace in which @euid applies
  * @pid: The process ID for the userspace program that sent the
@@ -272,7 +270,7 @@ int ecryptfs_process_helo(unsigned int transport, uid_t euid,
  * ecryptfs_exorcise_daemon - Destroy the daemon struct
  *
```

```
  * Must be called ceremoniously while in possession of
- * ecryptfs_daemon_hash_mux and the daemon's own mux.
+ * ecryptfs_daemon_hash_mux.
  */
 int ecryptfs_exorcise_daemon(struct ecryptfs_daemon *daemon)
 {
@@ -460,7 +458,7 @@ out:
 /**
  * ecryptfs_send_message_locked
  * @transport: The transport over which to send the message (i.e.,
- *             netlink)
+ *             miscdev)
  * @data: The data to send
  * @data_len: The length of data
  * @msg_ctx: The message context allocated for the send
@@ -496,14 +494,11 @@ ecryptfs_send_message_locked(unsigned int transport, char *data, int
data_len,
  mutex_unlock(&(*msg_ctx)->mux);
  mutex_unlock(&ecryptfs_msg_ctx_lists_mux);
  switch (transport) {
- case ECRYPTFS_TRANSPORT_NETLINK:
-  rc = ecryptfs_send_netlink(data, data_len, *msg_ctx, msg_type,
-         0, daemon->pid);
-  break;
  case ECRYPTFS_TRANSPORT_MISCDEV:
   rc = ecryptfs_send_miscdev(data, data_len, *msg_ctx, msg_type,
         0, daemon);
   break;
+ case ECRYPTFS_TRANSPORT_NETLINK:
  case ECRYPTFS_TRANSPORT_CONNECTOR:
  case ECRYPTFS_TRANSPORT_RELAYFS:
  default:
@@ -519,7 +514,7 @@ out:
 /**
  * ecryptfs_send_message
  * @transport: The transport over which to send the message (i.e.,
- *             netlink)
+ *             miscdev)
  * @data: The data to send
  * @data_len: The length of data
  * @msg_ctx: The message context allocated for the send
@@ -632,16 +627,12 @@ int ecryptfs_init_messaging(unsigned int transport)
  }
  mutex_unlock(&ecryptfs_msg_ctx_lists_mux);
  switch(transport) {
- case ECRYPTFS_TRANSPORT_NETLINK:
-  rc = ecryptfs_init_netlink();
-  if (rc)
```

```
-   ecryptfs_release_messaging(transport);
-  break;
   case ECRYPTFS_TRANSPORT_MISCDEV:
    rc = ecryptfs_init_ecryptfs_miscdev();
    if (rc)
     ecryptfs_release_messaging(transport);
    break;
+ case ECRYPTFS_TRANSPORT_NETLINK:
   case ECRYPTFS_TRANSPORT_CONNECTOR:
   case ECRYPTFS_TRANSPORT_RELAYFS:
   default:
@@ -691,12 +682,10 @@ void ecryptfs_release_messaging(unsigned int transport)
    mutex_unlock(&ecryptfs_daemon_hash_mux);
   }
   switch(transport) {
- case ECRYPTFS_TRANSPORT_NETLINK:
-  ecryptfs_release_netlink();
-  break;
   case ECRYPTFS_TRANSPORT_MISCDEV:
    ecryptfs_destroy_ecryptfs_miscdev();
    break;
+ case ECRYPTFS_TRANSPORT_NETLINK:
   case ECRYPTFS_TRANSPORT_CONNECTOR:
   case ECRYPTFS_TRANSPORT_RELAYFS:
   default:
diff --git a/fs/ecryptfs/netlink.c b/fs/ecryptfs/netlink.c
deleted file mode 100644
index e0abad6..0000000
--- a/fs/ecryptfs/netlink.c
+++ /dev/null
@@ -1,249 +0,0 @@
-/**
- * eCryptfs: Linux filesystem encryption layer
- *
- * Copyright (C) 2004-2006 International Business Machines Corp.
- *   Author(s): Michael A. Halcrow <mhalcrow@us.ibm.com>
- *  Tyler Hicks <tyhicks@ou.edu>
- *
- * This program is free software; you can redistribute it and/or
- * modify it under the terms of the GNU General Public License version
- * 2 as published by the Free Software Foundation.
- *
- * This program is distributed in the hope that it will be useful, but
- * WITHOUT ANY WARRANTY; without even the implied warranty of
- * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
- * General Public License for more details.
- *
- * You should have received a copy of the GNU General Public License
```

```
- * along with this program; if not, write to the Free Software
- * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
- * 02111-1307, USA.
- */
-
-#include <net/sock.h>
-#include <linux/hash.h>
-#include <linux/random.h>
-#include "ecryptfs_kernel.h"
-
-static struct sock *ecryptfs_nl_sock;
-
-/**
- * ecryptfs_send_netlink
- * @data: The data to include as the payload
- * @data_len: The byte count of the data
- * @msg_ctx: The netlink context that will be used to handle the
- *           response message
- * @msg_type: The type of netlink message to send
- * @msg_flags: The flags to include in the netlink header
- * @daemon_pid: The process id of the daemon to send the message to
- *
- * Sends the data to the specified daemon pid and uses the netlink
- * context element to store the data needed for validation upon
- * receiving the response.  The data and the netlink context can be
- * null if just sending a netlink header is sufficient.  Returns zero
- * upon sending the message; non-zero upon error.
- */
-int ecryptfs_send_netlink(char *data, int data_len,
-      struct ecryptfs_msg_ctx *msg_ctx, u8 msg_type,
-      u16 msg_flags, struct pid *daemon_pid)
-{
- struct sk_buff *skb;
- struct nlmsghdr *nlh;
- struct ecryptfs_message *msg;
- size_t payload_len;
- int rc;
-
- payload_len = ((data && data_len) ? (sizeof(*msg) + data_len) : 0);
- skb = alloc_skb(NLMSG_SPACE(payload_len), GFP_KERNEL);
- if (!skb) {
-  rc = -ENOMEM;
-  ecryptfs_printk(KERN_ERR, "Failed to allocate socket buffer\n");
-  goto out;
- }
- nlh = NLMSG_PUT(skb, pid_nr(daemon_pid), msg_ctx ? msg_ctx->counter : 0,
-    msg_type, payload_len);
- nlh->nlmsg_flags = msg_flags;
```

```
- if (msg_ctx && payload_len) {
-  msg = (struct ecryptfs_message *)NLMSG_DATA(nlh);
-  msg->index = msg_ctx->index;
-  msg->data_len = data_len;
-  memcpy(msg->data, data, data_len);
- }
- rc = netlink_unicast(ecryptfs_nl_sock, skb, pid_nr(daemon_pid), 0);
- if (rc < 0) {
-  ecryptfs_printk(KERN_ERR, "Failed to send eCryptfs netlink "
-    "message; rc = [%d]\n", rc);
-  goto out;
- }
- rc = 0;
- goto out;
-nlmsg_failure:
- rc = -EMSGSIZE;
- kfree_skb(skb);
-out:
- return rc;
-}
-
-/**
- * ecryptfs_process_nl_reponse
- * @skb: The socket buffer containing the netlink message of state
- *       RESPONSE
- *
- * Processes a response message after sending a operation request to
- * userspace.  Attempts to assign the msg to a netlink context element
- * at the index specified in the msg.  The sk_buff and nlmsghdr must
- * be validated before this function. Returns zero upon delivery to
- * desired context element; non-zero upon delivery failure or error.
- */
-static int ecryptfs_process_nl_response(struct sk_buff *skb)
-{
- struct nlmsghdr *nlh = nlmsg_hdr(skb);
- struct ecryptfs_message *msg = NLMSG_DATA(nlh);
- struct pid *pid;
- int rc;
-
- if (skb->len - NLMSG_HDRLEN - sizeof(*msg) != msg->data_len) {
-  rc = -EINVAL;
-  ecryptfs_printk(KERN_ERR, "Received netlink message with "
-    "incorrectly specified data length\n");
-  goto out;
- }
- pid = find_get_pid(NETLINK_CREDS(skb)->pid);
- rc = ecryptfs_process_response(msg, NETLINK_CREDS(skb)->uid, NULL,
-           pid, nlh->nlmsg_seq);
```

```
- put_pid(pid);
- if (rc)
-  printk(KERN_ERR
-         "Error processing response message; rc = [%d]\n", rc);
-out:
- return rc;
-}
-
-/**
- * ecryptfs_process_nl_helo
- * @skb: The socket buffer containing the nlmsghdr in HELO state
- *
- * Gets uid and pid of the skb and adds the values to the daemon id
- * hash. Returns zero after adding a new daemon id to the hash list;
- * non-zero otherwise.
- */
-static int ecryptfs_process_nl_helo(struct sk_buff *skb)
-{
- struct pid *pid;
- int rc;
-
- pid = find_get_pid(NETLINK_CREDS(skb)->pid);
- rc = ecryptfs_process_helo(ECRYPTFS_TRANSPORT_NETLINK,
-        NETLINK_CREDS(skb)->uid, NULL, pid);
- put_pid(pid);
- if (rc)
-  printk(KERN_WARNING "Error processing HELO; rc = [%d]\n", rc);
- return rc;
-}
-
-/**
- * ecryptfs_process_nl_quit
- * @skb: The socket buffer containing the nlmsghdr in QUIT state
- *
- * Gets uid and pid of the skb and deletes the corresponding daemon
- * id, if it is the registered that is requesting the
- * deletion. Returns zero after deleting the desired daemon id;
- * non-zero otherwise.
- */
-static int ecryptfs_process_nl_quit(struct sk_buff *skb)
-{
- struct pid *pid;
- int rc;
-
- pid = find_get_pid(NETLINK_CREDS(skb)->pid);
- rc = ecryptfs_process_quit(NETLINK_CREDS(skb)->uid, NULL, pid);
- put_pid(pid);
- if (rc)
```

```
-  printk(KERN_WARNING
-         "Error processing QUIT message; rc = [%d]\n", rc);
- return rc;
-}
-
-/**
- * ecryptfs_receive_nl_message
- *
- * Callback function called by netlink system when a message arrives.
- * If the message looks to be valid, then an attempt is made to assign
- * it to its desired netlink context element and wake up the process
- * that is waiting for a response.
- */
-static void ecryptfs_receive_nl_message(struct sk_buff *skb)
-{
- struct nlmsghdr *nlh;
-
- nlh = nlmsg_hdr(skb);
- if (!NLMSG_OK(nlh, skb->len)) {
-  ecryptfs_printk(KERN_ERR, "Received corrupt netlink "
-    "message\n");
-  goto free;
- }
- switch (nlh->nlmsg_type) {
-  case ECRYPTFS_MSG_RESPONSE:
-   if (ecryptfs_process_nl_response(skb)) {
-    ecryptfs_printk(KERN_WARNING, "Failed to "
-      "deliver netlink response to "
-      "requesting operation\n");
-   }
-   break;
-  case ECRYPTFS_MSG_HELO:
-   if (ecryptfs_process_nl_helo(skb)) {
-    ecryptfs_printk(KERN_WARNING, "Failed to "
-      "fulfill HELO request\n");
-   }
-   break;
-  case ECRYPTFS_MSG_QUIT:
-   if (ecryptfs_process_nl_quit(skb)) {
-    ecryptfs_printk(KERN_WARNING, "Failed to "
-      "fulfill QUIT request\n");
-   }
-   break;
-  default:
-   ecryptfs_printk(KERN_WARNING, "Dropping netlink "
-     "message of unrecognized type [%d]\n",
-     nlh->nlmsg_type);
-   break;
```

```
- }
-free:
- kfree_skb(skb);
-}
-
-/**
- * ecryptfs_init_netlink
- *
- * Initializes the daemon id hash list, netlink context array, and
- * necessary locks.  Returns zero upon success; non-zero upon error.
- */
-int ecryptfs_init_netlink(void)
-{
- int rc;
-
- ecryptfs_nl_sock = netlink_kernel_create(&init_net, NETLINK_ECRYPTFS, 0,
-       ecryptfs_receive_nl_message,
-       NULL, THIS_MODULE);
- if (!ecryptfs_nl_sock) {
-  rc = -EIO;
-  ecryptfs_printk(KERN_ERR, "Failed to create netlink socket\n");
-  goto out;
- }
- ecryptfs_nl_sock->sk_sndtimeo = ECRYPTFS_DEFAULT_SEND_TIMEOUT;
- rc = 0;
-out:
- return rc;
-}
-
-/**
- * ecryptfs_release_netlink
- *
- * Frees all memory used by the netlink context array and releases the
- * netlink socket.
- */
-void ecryptfs_release_netlink(void)
-{
- netlink_kernel_release(ecryptfs_nl_sock);
- ecryptfs_nl_sock = NULL;
-}
--
1.5.1.6
```