Subject: Re: [PATCH 1/2] eCryptfs: Introduce device handle for userspace daemon communications
Posted by akpm on Tue, 15 Apr 2008 21:04:53 GMT
View Forum Message <> Reply to Message

On Tue, 15 Apr 2008 15:23:13 -0500
Michael Halcrow <mhalcrow@us.ibm.com> wrote:

> Functions to facilitate reading and writing to the eCryptfs
> miscellaneous device handle. This will replace the netlink interface
> as the preferred mechanism for communicating with the userspace
> eCryptfs daemon.
>
> Each user has his own daemon, which registers itself by opening the
> eCryptfs device handle. Only one daemon per euid may be registered at
> any given time. The eCryptfs module sends a message to a daemon by
> adding its message to the daemon's outgoing message queue. The daemon
> reads the device handle to get the oldest message off the queue.
>
> Incoming messages from the userspace daemon are immediately
> handled. If the message is a response, then the corresponding process
> that is blocked waiting for the response is awakened.
>

This is a drastic change, but the changelog doesn't tell us why it is being
made!

> ...
> + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> + if (daemon->pid != current->pid) {
> + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> + BUG_ON(current->euid != daemon->euid);
> + BUG_ON(current->pid != daemon->pid);

This code uses pids and uids all over the place. Will it operate correctly
in a containerised environment?

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

Subject: Re: [PATCH 1/2] eCryptfs: Introduce device handle for userspace daemon communications
Posted by serue on Tue, 15 Apr 2008 21:34:02 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Tue, 15 Apr 2008 15:23:13 -0500
> Michael Halcrow <mhalcrow@us.ibm.com> wrote:
>
> > Functions to facilitate reading and writing to the eCryptfs
> > miscellaneous device handle. This will replace the netlink interface
> > as the preferred mechanism for communicating with the userspace
> > eCryptfs daemon.
> >
> > Each user has his own daemon, which registers itself by opening the
> > eCryptfs device handle. Only one daemon per euid may be registered at
> > any given time. The eCryptfs module sends a message to a daemon by
> > adding its message to the daemon's outgoing message queue. The daemon
> > reads the device handle to get the oldest message off the queue.
> >
> > Incoming messages from the userspace daemon are immediately
> > handled. If the message is a response, then the corresponding process
> > that is blocked waiting for the response is awakened.
> >
>
> This is a drastic change, but the changelog doesn't tell us why it is being
> made!
>
> > ...
> > + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> > + if (daemon->pid != current->pid) {
> > + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> > + BUG_ON(current->euid != daemon->euid);
> > + BUG_ON(current->pid != daemon->pid);
>
> This code uses pids and uids all over the place. Will it operate correctly
> in a containerised environment?

Thanks Andrew.

Mike, the pid_t definately needs to be replaced with a struct pid.

As for the euid, it'd be best if you also compared the user_namespace *
to make sure we support one ecryptfs deamon per user namespace.

thanks,
-serge

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

Subject: Re: [Ecryptfs-devel] [PATCH 1/2] eCryptfs: Introduce device handle for userspace daemon communicatio
Posted by Michael Halcrow on Tue, 15 Apr 2008 22:47:50 GMT
View Forum Message <> Reply to Message

On Tue, Apr 15, 2008 at 02:04:53PM -0700, Andrew Morton wrote:
> On Tue, 15 Apr 2008 15:23:13 -0500
> Michael Halcrow <mhalcrow@us.ibm.com> wrote:
>
> > Functions to facilitate reading and writing to the eCryptfs
> > miscellaneous device handle. This will replace the netlink interface
> > as the preferred mechanism for communicating with the userspace
> > eCryptfs daemon.
> >
> > Each user has his own daemon, which registers itself by opening the
> > eCryptfs device handle. Only one daemon per euid may be registered at
> > any given time. The eCryptfs module sends a message to a daemon by
> > adding its message to the daemon's outgoing message queue. The daemon
> > reads the device handle to get the oldest message off the queue.
> >
> > Incoming messages from the userspace daemon are immediately
> > handled. If the message is a response, then the corresponding process
> > that is blocked waiting for the response is awakened.
>
> This is a drastic change, but the changelog doesn't tell us why it
> is being made!

This resurrects the question from 2006:

On Thu, Aug 24, 2006 at 08:54:19PM -0700, Andrew Morton wrote:
> - _why_ does it use netlink?

I responded:

> Netlink provides the transport mechanism that would minimize the
> complexity of the implementation, given that we can have multiple
> daemons (one per user).

A regular device file was my real preference from the get-go, but I went with netlink at the time because I thought it would be less complex for managing send queues (i.e., just do a unicast and move on). It turns out that we do not really get that much complexity reduction with netlink, and netlink is more heavyweight than a device handle.

In addition, the netlink interface to eCryptfs has been broken since 2.6.24. I am assuming this is a bug in how eCryptfs uses netlink, since the other in-kernel users of netlink do not seem to be having any problems. I have had one report of a user successfully using

eCryptfs with netlink on 2.6.24, but for my own systems, when starting the userspace daemon, the initial helo message sent to the eCryptfs kernel module results in an oops right off the bat. I spent some time looking at it, but I have not yet found the cause. The netlink interface breaking gave me the motivation to just finish my patch to migrate to a regular device handle. If I cannot find out soon why the netlink interface in eCryptfs broke, I am likely to just send a patch to disable it in 2.6.24 and 2.6.25. I would like the device handle to be the preferred means of communicating with the userspace daemon from 2.6.26 on forward.

Mike

Subject: Re: [Ecryptfs-devel] [PATCH 1/2] eCryptfs: Introduce device handle for userspace daemon communicatio
Posted by Michael Halcrow on Tue, 15 Apr 2008 23:30:03 GMT
View Forum Message <> Reply to Message

On Tue, Apr 15, 2008 at 05:47:50PM -0500, Michael Halcrow wrote:
> If I cannot find out soon why the netlink interface in eCryptfs
> broke, I am likely to just send a patch to disable it in 2.6.24 and
> 2.6.25.

This patch yanks key module support out of eCryptfs. In the event that the netlink bug with eCryptfs does not get resolved soon, I intend for this to apply to point releases of 2.6.24 and 2.6.25.

This patch is only a feature-disabling workaround and conflicts with the patchset that introduces the device handle mechanism for communicating with the userspace key module daemon. Future kernel versions will use a regular device handle for communications with the userspace daemon rather than the netlink interface.

Signed-off-by: Michael Halcrow <mhalcrow@us.ibm.com>
---
 fs/ecryptfs/ecryptfs_kernel.h | 1 -
 fs/ecryptfs/keystore.c | 422 ++--
 fs/ecryptfs/main.c | 11 +-
 fs/ecryptfs/messaging.c | 6 -
 4 files changed, 13 insertions(+), 427 deletions(-)

diff --git a/fs/ecryptfs/ecryptfs_kernel.h b/fs/ecryptfs/ecryptfs_kernel.h
index 5007f78..abff330 100644

```
--- a/fs/ecryptfs/ecryptfs_kernel.h
+++ b/fs/ecryptfs/ecryptfs_kernel.h
@@ -51,7 +51,6 @@
 #define ECRYPTFS_VERSIONING_MULTKEY             0x00000020
 #define ECRYPTFS_VERSIONING_MASK (ECRYPTFS_VERSIONING_PASSPHRASE \
       | ECRYPTFS_VERSIONING_PLAINTEXT_PASSTHROUGH \
-      | ECRYPTFS_VERSIONING_PUBKEY \
       | ECRYPTFS_VERSIONING_XATTR \
       | ECRYPTFS_VERSIONING_MULTKEY)
 #define ECRYPTFS_MAX_PASSWORD_LENGTH 64
diff --git a/fs/ecryptfs/keystore.c b/fs/ecryptfs/keystore.c
index 682b1b2..8f65054 100644
--- a/fs/ecryptfs/keystore.c
+++ b/fs/ecryptfs/keystore.c
@@ -273,129 +273,6 @@ out:
  return rc;
 }

-
-static int
-write_tag_66_packet(char *signature, u8 cipher_code,
-      struct ecryptfs_crypt_stat *crypt_stat, char **packet,
-      size_t *packet_len)
-{
- size_t i = 0;
- size_t j;
- size_t data_len;
- size_t checksum = 0;
- size_t packet_size_len;
- char *message;
- int rc;
-
- /*
-  *              ***** TAG 66 Packet Format *****
-  *         | Content Type             | 1 byte       |
-  *         | Key Identifier Size      | 1 or 2 bytes |
-  *         | Key Identifier           | arbitrary    |
-  *         | File Encryption Key Size | 1 or 2 bytes |
-  *         | File Encryption Key      | arbitrary    |
-  */
- data_len = (5 + ECRYPTFS_SIG_SIZE_HEX + crypt_stat->key_size);
- *packet = kmalloc(data_len, GFP_KERNEL);
- message = *packet;
- if (!message) {
-  ecryptfs_printk(KERN_ERR, "Unable to allocate memory\n");
-  rc = -ENOMEM;
-  goto out;
- }
```

```
- message[i++] = ECRYPTFS_TAG_66_PACKET_TYPE;
- rc = write_packet_length(&message[i], ECRYPTFS_SIG_SIZE_HEX,
-     &packet_size_len);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Error generating tag 66 packet "
-    "header; cannot generate packet length\n");
-  goto out;
- }
- i += packet_size_len;
- memcpy(&message[i], signature, ECRYPTFS_SIG_SIZE_HEX);
- i += ECRYPTFS_SIG_SIZE_HEX;
- /* The encrypted key includes 1 byte cipher code and 2 byte checksum */
- rc = write_packet_length(&message[i], crypt_stat->key_size + 3,
-     &packet_size_len);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Error generating tag 66 packet "
-    "header; cannot generate packet length\n");
-  goto out;
- }
- i += packet_size_len;
- message[i++] = cipher_code;
- memcpy(&message[i], crypt_stat->key, crypt_stat->key_size);
- i += crypt_stat->key_size;
- for (j = 0; j < crypt_stat->key_size; j++)
-  checksum += crypt_stat->key[j];
- message[i++] = (checksum / 256) % 256;
- message[i++] = (checksum % 256);
- *packet_len = i;
-out:
- return rc;
-}
-
-static int
-parse_tag_67_packet(struct ecryptfs_key_record *key_rec,
-      struct ecryptfs_message *msg)
-{
- size_t i = 0;
- char *data;
- size_t data_len;
- size_t message_len;
- int rc;
-
- /*
-  *             ***** TAG 65 Packet Format *****
-  *    | Content Type                       | 1 byte       |
-  *    | Status Indicator                   | 1 byte       |
-  *    | Encrypted File Encryption Key Size | 1 or 2 bytes |
-  *    | Encrypted File Encryption Key      | arbitrary    |
```

```
-  */
- message_len = msg->data_len;
- data = msg->data;
- /* verify that everything through the encrypted FEK size is present */
- if (message_len < 4) {
-  rc = -EIO;
-  goto out;
- }
- if (data[i++] != ECRYPTFS_TAG_67_PACKET_TYPE) {
-  ecryptfs_printk(KERN_ERR, "Type should be ECRYPTFS_TAG_67\n");
-  rc = -EIO;
-  goto out;
- }
- if (data[i++]) {
-  ecryptfs_printk(KERN_ERR, "Status indicator has non zero value"
-    " [%d]\n", data[i-1]);
-  rc = -EIO;
-  goto out;
- }
- rc = parse_packet_length(&data[i], &key_rec->enc_key_size, &data_len);
- if (rc) {
-  ecryptfs_printk(KERN_WARNING, "Error parsing packet length; "
-    "rc = [%d]\n", rc);
-  goto out;
- }
- i += data_len;
- if (message_len < (i + key_rec->enc_key_size)) {
-  ecryptfs_printk(KERN_ERR, "message_len [%d]; max len is [%d]\n",
-    message_len, (i + key_rec->enc_key_size));
-  rc = -EIO;
-  goto out;
- }
- if (key_rec->enc_key_size > ECRYPTFS_MAX_ENCRYPTED_KEY_BYTES) {
-  ecryptfs_printk(KERN_ERR, "Encrypted key_size [%d] larger than "
-    "the maximum key size [%d]\n",
-    key_rec->enc_key_size,
-    ECRYPTFS_MAX_ENCRYPTED_KEY_BYTES);
-  rc = -EIO;
-  goto out;
- }
- memcpy(key_rec->enc_key, &data[i], key_rec->enc_key_size);
-out:
- return rc;
-}
-
 static int
 ecryptfs_get_auth_tok_sig(char **sig, struct ecryptfs_auth_tok *auth_tok)
 {
```

```
@@ -407,11 +284,11 @@ ecryptfs_get_auth_tok_sig(char **sig, struct ecryptfs_auth_tok
*auth_tok)
   (*sig) = auth_tok->token.password.signature;
   break;
  case ECRYPTFS_PRIVATE_KEY:
-  (*sig) = auth_tok->token.private_key.signature;
-  break;
+  printk(KERN_ERR "%s: Key module functionality disabled for "
+         "this kernel\n", __func__);
  default:
-  printk(KERN_ERR "Cannot get sig for auth_tok of type [%d]\n",
-         auth_tok->token_type);
+  printk(KERN_ERR "Cannot get valid sig for auth_tok of type "
+         "[%d]\n", auth_tok->token_type);
   rc = -EINVAL;
  }
  return rc;
@@ -506,137 +383,6 @@ static void wipe_auth_tok_list(struct list_head *auth_tok_list_head)
 struct kmem_cache *ecryptfs_auth_tok_list_item_cache;

 /**
- * parse_tag_1_packet
- * @crypt_stat: The cryptographic context to modify based on packet contents
- * @data: The raw bytes of the packet.
- * @auth_tok_list: eCryptfs parses packets into authentication tokens;
- *                 a new authentication token will be placed at the
- *                 end of this list for this packet.
- * @new_auth_tok: Pointer to a pointer to memory that this function
- *                allocates; sets the memory address of the pointer to
- *                NULL on error. This object is added to the
- *                auth_tok_list.
- * @packet_size: This function writes the size of the parsed packet
- *               into this memory location; zero on error.
- * @max_packet_size: The maximum allowable packet size
- *
- * Returns zero on success; non-zero on error.
- */
-static int
-parse_tag_1_packet(struct ecryptfs_crypt_stat *crypt_stat,
-     unsigned char *data, struct list_head *auth_tok_list,
-     struct ecryptfs_auth_tok **new_auth_tok,
-     size_t *packet_size, size_t max_packet_size)
-{
- size_t body_size;
- struct ecryptfs_auth_tok_list_item *auth_tok_list_item;
- size_t length_size;
- int rc = 0;
-
```

```
- (*packet_size) = 0;
- (*new_auth_tok) = NULL;
- /**
-  * This format is inspired by OpenPGP; see RFC 2440
-  * packet tag 1
-  *
-  * Tag 1 identifier (1 byte)
-  * Max Tag 1 packet size (max 3 bytes)
-  * Version (1 byte)
-  * Key identifier (8 bytes; ECRYPTFS_SIG_SIZE)
-  * Cipher identifier (1 byte)
-  * Encrypted key size (arbitrary)
-  *
-  * 12 bytes minimum packet size
-  */
- if (unlikely(max_packet_size < 12)) {
-  printk(KERN_ERR "Invalid max packet size; must be >=12\n");
-  rc = -EINVAL;
-  goto out;
- }
- if (data[(*packet_size)++] != ECRYPTFS_TAG_1_PACKET_TYPE) {
-  printk(KERN_ERR "Enter w/ first byte != 0x%.2x\n",
-         ECRYPTFS_TAG_1_PACKET_TYPE);
-  rc = -EINVAL;
-  goto out;
- }
- /* Released: wipe_auth_tok_list called in ecryptfs_parse_packet_set or
-  * at end of function upon failure */
- auth_tok_list_item =
-  kmem_cache_zalloc(ecryptfs_auth_tok_list_item_cache,
-      GFP_KERNEL);
- if (!auth_tok_list_item) {
-  printk(KERN_ERR "Unable to allocate memory\n");
-  rc = -ENOMEM;
-  goto out;
- }
- (*new_auth_tok) = &auth_tok_list_item->auth_tok;
- rc = parse_packet_length(&data[(*packet_size)], &body_size,
-     &length_size);
- if (rc) {
-  printk(KERN_WARNING "Error parsing packet length; "
-         "rc = [%d]\n", rc);
-  goto out_free;
- }
- if (unlikely(body_size < (ECRYPTFS_SIG_SIZE + 2))) {
-  printk(KERN_WARNING "Invalid body size ([%td])\n", body_size);
-  rc = -EINVAL;
-  goto out_free;
```

```
- }
- (*packet_size) += length_size;
- if (unlikely((*packet_size) + body_size > max_packet_size)) {
-  printk(KERN_WARNING "Packet size exceeds max\n");
-  rc = -EINVAL;
-  goto out_free;
- }
- if (unlikely(data[(*packet_size)++] != 0x03)) {
-  printk(KERN_WARNING "Unknown version number [%d]\n",
-         data[(*packet_size) - 1]);
-  rc = -EINVAL;
-  goto out_free;
- }
- ecryptfs_to_hex((*new_auth_tok)->token.private_key.signature,
-   &data[(*packet_size)], ECRYPTFS_SIG_SIZE);
- *packet_size += ECRYPTFS_SIG_SIZE;
- /* This byte is skipped because the kernel does not need to
-  * know which public key encryption algorithm was used */
- (*packet_size)++;
- (*new_auth_tok)->session_key.encrypted_key_size =
-  body_size - (ECRYPTFS_SIG_SIZE + 2);
- if ((*new_auth_tok)->session_key.encrypted_key_size
-     > ECRYPTFS_MAX_ENCRYPTED_KEY_BYTES) {
-  printk(KERN_WARNING "Tag 1 packet contains key larger "
-         "than ECRYPTFS_MAX_ENCRYPTED_KEY_BYTES");
-  rc = -EINVAL;
-  goto out;
- }
- memcpy((*new_auth_tok)->session_key.encrypted_key,
-        &data[(*packet_size)], (body_size - (ECRYPTFS_SIG_SIZE + 2)));
- (*packet_size) += (*new_auth_tok)->session_key.encrypted_key_size;
- (*new_auth_tok)->session_key.flags &=
-  ~ECRYPTFS_CONTAINS_DECRYPTED_KEY;
- (*new_auth_tok)->session_key.flags |=
-  ECRYPTFS_CONTAINS_ENCRYPTED_KEY;
- (*new_auth_tok)->token_type = ECRYPTFS_PRIVATE_KEY;
- (*new_auth_tok)->flags = 0;
- (*new_auth_tok)->session_key.flags &=
-  ~(ECRYPTFS_USERSPACE_SHOULD_TRY_TO_DECRYPT);
- (*new_auth_tok)->session_key.flags &=
-  ~(ECRYPTFS_USERSPACE_SHOULD_TRY_TO_ENCRYPT);
- list_add(&auth_tok_list_item->list, auth_tok_list);
- goto out;
-out_free:
- (*new_auth_tok) = NULL;
- memset(auth_tok_list_item, 0,
-        sizeof(struct ecryptfs_auth_tok_list_item));
- kmem_cache_free(ecryptfs_auth_tok_list_item_cache,
```

```
-   auth_tok_list_item);
-out:
- if (rc)
-  (*packet_size) = 0;
- return rc;
-}
-
-/**
  * parse_tag_3_packet
  * @crypt_stat: The cryptographic context to modify based on packet
  *              contents.
@@ -1197,18 +943,10 @@ int ecryptfs_parse_packet_set(struct ecryptfs_crypt_stat *crypt_stat,
    crypt_stat->flags |= ECRYPTFS_ENCRYPTED;
    break;
   case ECRYPTFS_TAG_1_PACKET_TYPE:
-   rc = parse_tag_1_packet(crypt_stat,
-      (unsigned char *)&src[i],
-      &auth_tok_list, &new_auth_tok,
-      &packet_size, max_packet_size);
-   if (rc) {
-    ecryptfs_printk(KERN_ERR, "Error parsing "
-      "tag 1 packet\n");
-    rc = -EIO;
-    goto out_wipe_list;
-   }
-   i += packet_size;
-   crypt_stat->flags |= ECRYPTFS_ENCRYPTED;
+   rc = -EIO;
+   printk(KERN_ERR "%s: No public key packet support in "
+          "this kernel release\n", __func__);
+   goto out_wipe_list;
    break;
   case ECRYPTFS_TAG_11_PACKET_TYPE:
    ecryptfs_printk(KERN_WARNING, "Invalid packet set "
@@ -1322,137 +1060,6 @@ out:
  return rc;
 }

-static int
-pki_encrypt_session_key(struct ecryptfs_auth_tok *auth_tok,
-   struct ecryptfs_crypt_stat *crypt_stat,
-   struct ecryptfs_key_record *key_rec)
-{
- struct ecryptfs_msg_ctx *msg_ctx = NULL;
- char *netlink_payload;
- size_t netlink_payload_length;
- struct ecryptfs_message *msg;
- int rc;
```

```
-
- rc = write_tag_66_packet(auth_tok->token.private_key.signature,
-     ecryptfs_code_for_cipher_string(crypt_stat),
-     crypt_stat, &netlink_payload,
-     &netlink_payload_length);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Error generating tag 66 packet\n");
-  goto out;
- }
- rc = ecryptfs_send_message(ecryptfs_transport, netlink_payload,
-       netlink_payload_length, &msg_ctx);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Error sending netlink message\n");
-  goto out;
- }
- rc = ecryptfs_wait_for_response(msg_ctx, &msg);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Failed to receive tag 67 packet "
-    "from the user space daemon\n");
-  rc = -EIO;
-  goto out;
- }
- rc = parse_tag_67_packet(key_rec, msg);
- if (rc)
-  ecryptfs_printk(KERN_ERR, "Error parsing tag 67 packet\n");
- kfree(msg);
-out:
- if (netlink_payload)
-  kfree(netlink_payload);
- return rc;
-}
-/**
- * write_tag_1_packet - Write an RFC2440-compatible tag 1 (public key) packet
- * @dest: Buffer into which to write the packet
- * @remaining_bytes: Maximum number of bytes that can be writtn
- * @auth_tok: The authentication token used for generating the tag 1 packet
- * @crypt_stat: The cryptographic context
- * @key_rec: The key record struct for the tag 1 packet
- * @packet_size: This function will write the number of bytes that end
- *               up constituting the packet; set to zero on error
- *
- * Returns zero on success; non-zero on error.
- */
-static int
-write_tag_1_packet(char *dest, size_t *remaining_bytes,
-     struct ecryptfs_auth_tok *auth_tok,
-     struct ecryptfs_crypt_stat *crypt_stat,
-     struct ecryptfs_key_record *key_rec, size_t *packet_size)
```

```
-{
- size_t i;
- size_t encrypted_session_key_valid = 0;
- size_t packet_size_length;
- size_t max_packet_size;
- int rc = 0;
-
- (*packet_size) = 0;
- ecryptfs_from_hex(key_rec->sig, auth_tok->token.private_key.signature,
-     ECRYPTFS_SIG_SIZE);
- encrypted_session_key_valid = 0;
- for (i = 0; i < crypt_stat->key_size; i++)
-  encrypted_session_key_valid |=
-   auth_tok->session_key.encrypted_key[i];
- if (encrypted_session_key_valid) {
-  memcpy(key_rec->enc_key,
-         auth_tok->session_key.encrypted_key,
-         auth_tok->session_key.encrypted_key_size);
-  goto encrypted_session_key_set;
- }
- if (auth_tok->session_key.encrypted_key_size == 0)
-  auth_tok->session_key.encrypted_key_size =
-   auth_tok->token.private_key.key_size;
- rc = pki_encrypt_session_key(auth_tok, crypt_stat, key_rec);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Failed to encrypt session key "
-    "via a pki");
-  goto out;
- }
- if (ecryptfs_verbosity > 0) {
-  ecryptfs_printk(KERN_DEBUG, "Encrypted key:\n");
-  ecryptfs_dump_hex(key_rec->enc_key, key_rec->enc_key_size);
- }
-encrypted_session_key_set:
- /* This format is inspired by OpenPGP; see RFC 2440
-  * packet tag 1 */
- max_packet_size = (1                        /* Tag 1 identifier */
-      + 3                    /* Max Tag 1 packet size */
-      + 1                    /* Version */
-      + ECRYPTFS_SIG_SIZE       /* Key identifier */
-      + 1                    /* Cipher identifier */
-      + key_rec->enc_key_size); /* Encrypted key size */
- if (max_packet_size > (*remaining_bytes)) {
-  printk(KERN_ERR "Packet length larger than maximum allowable; "
-         "need up to [%td] bytes, but there are only [%td] "
-         "available\n", max_packet_size, (*remaining_bytes));
-  rc = -EINVAL;
-  goto out;
```

```
- }
- dest[(*packet_size)++] = ECRYPTFS_TAG_1_PACKET_TYPE;
- rc = write_packet_length(&dest[(*packet_size)], (max_packet_size - 4),
-     &packet_size_length);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Error generating tag 1 packet "
-    "header; cannot generate packet length\n");
-  goto out;
- }
- (*packet_size) += packet_size_length;
- dest[(*packet_size)++] = 0x03; /* version 3 */
- memcpy(&dest[(*packet_size)], key_rec->sig, ECRYPTFS_SIG_SIZE);
- (*packet_size) += ECRYPTFS_SIG_SIZE;
- dest[(*packet_size)++] = RFC2440_CIPHER_RSA;
- memcpy(&dest[(*packet_size)], key_rec->enc_key,
-        key_rec->enc_key_size);
- (*packet_size) += key_rec->enc_key_size;
-out:
- if (rc)
-  (*packet_size) = 0;
- else
-  (*remaining_bytes) -= (*packet_size);
- return rc;
-}
-
 /**
  * write_tag_11_packet
  * @dest: Target into which Tag 11 packet is to be written
@@ -1798,15 +1405,10 @@ ecryptfs_generate_key_packet_set(char *dest_base,
    }
    (*len) += written;
   } else if (auth_tok->token_type == ECRYPTFS_PRIVATE_KEY) {
-   rc = write_tag_1_packet(dest_base + (*len),
-      &max, auth_tok,
-      crypt_stat, key_rec, &written);
-   if (rc) {
-    ecryptfs_printk(KERN_WARNING, "Error "
-      "writing tag 1 packet\n");
-    goto out_free;
-   }
-   (*len) += written;
+   rc = -EIO;
+   printk(KERN_ERR "%s: No public key packet support in "
+          "this kernel release\n", __func__);
+   goto out_free;
   } else {
    ecryptfs_printk(KERN_WARNING, "Unsupported "
     "authentication token type\n");
```

```
diff --git a/fs/ecryptfs/main.c b/fs/ecryptfs/main.c
index d25ac95..5f03d63 100644
--- a/fs/ecryptfs/main.c
+++ b/fs/ecryptfs/main.c
@@ -795,25 +795,17 @@ static int __init ecryptfs_init(void)
   printk(KERN_ERR "sysfs registration failed\n");
   goto out_unregister_filesystem;
  }
- rc = ecryptfs_init_messaging(ecryptfs_transport);
- if (rc) {
-  ecryptfs_printk(KERN_ERR, "Failure occured while attempting to "
-    "initialize the eCryptfs netlink socket\n");
-  goto out_do_sysfs_unregistration;
- }
  rc = ecryptfs_init_crypto();
  if (rc) {
   printk(KERN_ERR "Failure whilst attempting to init crypto; "
          "rc = [%d]\n", rc);
-  goto out_release_messaging;
+  goto out_do_sysfs_unregistration;
  }
  if (ecryptfs_verbosity > 0)
   printk(KERN_CRIT "eCryptfs verbosity set to %d. Secret values "
    "will be written to the syslog!\n", ecryptfs_verbosity);

  goto out;
-out_release_messaging:
- ecryptfs_release_messaging(ecryptfs_transport);
 out_do_sysfs_unregistration:
  do_sysfs_unregistration();
 out_unregister_filesystem:
@@ -832,7 +824,6 @@ static void __exit ecryptfs_exit(void)
  if (rc)
   printk(KERN_ERR "Failure whilst attempting to destroy crypto; "
          "rc = [%d]\n", rc);
- ecryptfs_release_messaging(ecryptfs_transport);
  do_sysfs_unregistration();
  unregister_filesystem(&ecryptfs_fs_type);
  ecryptfs_free_kmem_caches();
diff --git a/fs/ecryptfs/messaging.c b/fs/ecryptfs/messaging.c
index 9cc2aec..c909dc3 100644
--- a/fs/ecryptfs/messaging.c
+++ b/fs/ecryptfs/messaging.c
@@ -460,10 +460,6 @@ 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_CONNECTOR:
  case ECRYPTFS_TRANSPORT_RELAYFS:
  default:
@@ -507,8 +503,6 @@ void ecryptfs_release_messaging(unsigned int transport)
  }
  switch(transport) {
  case ECRYPTFS_TRANSPORT_NETLINK:
-  ecryptfs_release_netlink();
-  break;
  case ECRYPTFS_TRANSPORT_CONNECTOR:
  case ECRYPTFS_TRANSPORT_RELAYFS:
  default:
--
1.5.3.6
```

Subject: [PATCH] eCryptfs: Make key module subsystem respect namespaces
Posted by Michael Halcrow on Wed, 16 Apr 2008 19:24:17 GMT
View Forum Message <> Reply to Message

On Tue, Apr 15, 2008 at 04:34:02PM -0500, Serge E. Hallyn wrote:
> Quoting Andrew Morton (akpm@linux-foundation.org):
> > On Tue, 15 Apr 2008 15:23:13 -0500
> > Michael Halcrow <mhalcrow@us.ibm.com> wrote:
> >
> > > ...
> > > + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> > > + if (daemon->pid != current->pid) {
> > > + rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
> > > + BUG_ON(current->euid != daemon->euid);
> > > + BUG_ON(current->pid != daemon->pid);
> >
> > This code uses pids and uids all over the place. Will it operate
> > correctly in a containerised environment?
>
> Thanks Andrew.
>
> Mike, the pid_t definately needs to be replaced with a struct pid.
>
> As for the euid, it'd be best if you also compared the user_namespace *

> to make sure we support one ecryptfs deamon per user namespace.

Make eCryptfs key module subsystem respect namespaces.

Since I will be removing the netlink interface in a future patch, I
just made changes to the netlink.c code so that it will not break the
build. With my recent patches, the kernel module currently defaults to
the device handle interface rather than the netlink interface.

Signed-off-by: Michael Halcrow <mhalcrow@us.ibm.com>

```
---
 fs/ecryptfs/ecryptfs_kernel.h |   25 +++++++++-----
 fs/ecryptfs/messaging.c       |   71 ++++++++++++++++++++++++++++-----------
 fs/ecryptfs/miscdev.c         |   68 ++++++++++++++++++++++++--------------
 fs/ecryptfs/netlink.c         |   25 +++++++++-----
 4 files changed, 126 insertions(+), 63 deletions(-)

diff --git a/fs/ecryptfs/ecryptfs_kernel.h b/fs/ecryptfs/ecryptfs_kernel.h
index 88b85f7..dc11431 100644
--- a/fs/ecryptfs/ecryptfs_kernel.h
+++ b/fs/ecryptfs/ecryptfs_kernel.h
@@ -34,6 +34,7 @@
 #include <linux/namei.h>
 #include <linux/scatterlist.h>
 #include <linux/hash.h>
+#include <linux/nsproxy.h>

 /* Version verification for shared data structures w/ userspace */
 #define ECRYPTFS_VERSION_MAJOR 0x00
@@ -410,8 +411,9 @@ struct ecryptfs_daemon {
 #define ECRYPTFS_DAEMON_MISCDEV_OPEN 0x00000008
  u32 flags;
  u32 num_queued_msg_ctx;
- pid_t pid;
+ struct pid *pid;
  uid_t euid;
+ struct user_namespace *user_ns;
  struct task_struct *task;
  struct mutex mux;
  struct list_head msg_ctx_out_queue;
@@ -610,10 +612,13 @@ int
 ecryptfs_setxattr(struct dentry *dentry, const char *name, const void *value,
     size_t size, int flags);
 int ecryptfs_read_xattr_region(char *page_virt, struct inode *ecryptfs_inode);
-int ecryptfs_process_helo(unsigned int transport, uid_t uid, pid_t pid);
-int ecryptfs_process_quit(uid_t uid, pid_t pid);
-int ecryptfs_process_response(struct ecryptfs_message *msg, uid_t uid,
-         pid_t pid, u32 seq);
```

```
+int ecryptfs_process_helo(unsigned int transport, uid_t euid,
+     struct user_namespace *user_ns, struct pid *pid);
+int ecryptfs_process_quit(uid_t euid, struct user_namespace *user_ns,
+     struct pid *pid);
+int ecryptfs_process_response(struct ecryptfs_message *msg, uid_t euid,
+         struct user_namespace *user_ns, struct pid *pid,
+         u32 seq);
 int ecryptfs_send_message(unsigned int transport, char *data, int data_len,
      struct ecryptfs_msg_ctx **msg_ctx);
 int ecryptfs_wait_for_response(struct ecryptfs_msg_ctx *msg_ctx,
@@ -623,13 +628,13 @@ 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, pid_t daemon_pid);
+     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, pid_t daemon_pid);
+       u16 msg_flags, struct pid *daemon_pid);
 int ecryptfs_init_connector(void);
 void ecryptfs_release_connector(void);
 void
@@ -672,7 +677,8 @@ int ecryptfs_read_lower_page_segment(struct page *page_for_ecryptfs,
          struct inode *ecryptfs_inode);
 struct page *ecryptfs_get_locked_page(struct file *file, loff_t index);
 int ecryptfs_exorcise_daemon(struct ecryptfs_daemon *daemon);
-int ecryptfs_find_daemon_by_euid(struct ecryptfs_daemon **daemon, uid_t euid);
+int ecryptfs_find_daemon_by_euid(struct ecryptfs_daemon **daemon, uid_t euid,
+     struct user_namespace *user_ns);
 int ecryptfs_parse_packet_length(unsigned char *data, size_t *size,
      size_t *length_size);
 int ecryptfs_write_packet_length(char *dest, size_t size,
@@ -684,6 +690,7 @@ int ecryptfs_send_miscdev(char *data, size_t data_size,
      u16 msg_flags, struct ecryptfs_daemon *daemon);
 void ecryptfs_msg_ctx_alloc_to_free(struct ecryptfs_msg_ctx *msg_ctx);
 int
-ecryptfs_spawn_daemon(struct ecryptfs_daemon **daemon, uid_t euid, pid_t pid);
+ecryptfs_spawn_daemon(struct ecryptfs_daemon **daemon, uid_t euid,
+         struct user_namespace *user_ns, struct pid *pid);

 #endif /* #ifndef ECRYPTFS_KERNEL_H */
diff --git a/fs/ecryptfs/messaging.c b/fs/ecryptfs/messaging.c
index e3f2e97..fad161b 100644
--- a/fs/ecryptfs/messaging.c
```

```
+++ b/fs/ecryptfs/messaging.c
@@ -103,6 +103,7 @@ void ecryptfs_msg_ctx_alloc_to_free(struct ecryptfs_msg_ctx *msg_ctx)
 /**
  * ecryptfs_find_daemon_by_euid
  * @euid: The effective user id which maps to the desired daemon id
+ * @user_ns: The namespace in which @euid applies
  * @daemon: If return value is zero, points to the desired daemon pointer
  *
  * Must be called with ecryptfs_daemon_hash_mux held.
@@ -111,7 +112,8 @@ void ecryptfs_msg_ctx_alloc_to_free(struct ecryptfs_msg_ctx *msg_ctx)
  *
  * Returns zero if the user id exists in the list; non-zero otherwise.
  */
-int ecryptfs_find_daemon_by_euid(struct ecryptfs_daemon **daemon, uid_t euid)
+int ecryptfs_find_daemon_by_euid(struct ecryptfs_daemon **daemon, uid_t euid,
+     struct user_namespace *user_ns)
 {
  struct hlist_node *elem;
  int rc;
@@ -119,7 +121,7 @@ int ecryptfs_find_daemon_by_euid(struct ecryptfs_daemon **daemon,
uid_t euid)
  hlist_for_each_entry(*daemon, elem,
         &ecryptfs_daemon_hash[ecryptfs_uid_hash(euid)],
         euid_chain) {
-  if ((*daemon)->euid == euid) {
+  if ((*daemon)->euid == euid && (*daemon)->user_ns == user_ns) {
    rc = 0;
    goto out;
   }
@@ -186,6 +188,7 @@ out:
  * ecryptfs_spawn_daemon - Create and initialize a new daemon struct
  * @daemon: Pointer to set to newly allocated daemon struct
  * @euid: Effective user id for the daemon
+ * @user_ns: The namespace in which @euid applies
  * @pid: Process id for the daemon
  *
  * Must be called ceremoniously while in possession of
@@ -194,7 +197,8 @@ out:
  * Returns zero on success; non-zero otherwise
  */
 int
-ecryptfs_spawn_daemon(struct ecryptfs_daemon **daemon, uid_t euid, pid_t pid)
+ecryptfs_spawn_daemon(struct ecryptfs_daemon **daemon, uid_t euid,
+        struct user_namespace *user_ns, struct pid *pid)
 {
  int rc = 0;

@@ -206,6 +210,7 @@ ecryptfs_spawn_daemon(struct ecryptfs_daemon **daemon, uid_t euid,
```

```
pid_t pid)
   goto out;
  }
  (*daemon)->euid = euid;
+ (*daemon)->user_ns = user_ns;
  (*daemon)->pid = pid;
  (*daemon)->task = current;
  mutex_init(&(*daemon)->mux);
@@ -222,6 +227,7 @@ out:
  * ecryptfs_process_helo
  * @transport: The underlying transport (netlink, 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
  *       message
  *
@@ -231,32 +237,33 @@ out:
  * Returns zero after adding a new daemon to the hash list;
  * non-zero otherwise.
  */
-int ecryptfs_process_helo(unsigned int transport, uid_t euid, pid_t pid)
+int ecryptfs_process_helo(unsigned int transport, uid_t euid,
+     struct user_namespace *user_ns, struct pid *pid)
 {
  struct ecryptfs_daemon *new_daemon;
  struct ecryptfs_daemon *old_daemon;
  int rc;

  mutex_lock(&ecryptfs_daemon_hash_mux);
- rc = ecryptfs_find_daemon_by_euid(&old_daemon, euid);
+ rc = ecryptfs_find_daemon_by_euid(&old_daemon, euid, user_ns);
  if (rc != 0) {
   printk(KERN_WARNING "Received request from user [%d] "
-         "to register daemon [%d]; unregistering daemon "
-         "[%d]\n", euid, pid, old_daemon->pid);
+         "to register daemon [0x%p]; unregistering daemon "
+         "[0x%p]\n", euid, pid, old_daemon->pid);
   rc = ecryptfs_send_raw_message(transport, ECRYPTFS_MSG_QUIT,
            old_daemon);
   if (rc)
    printk(KERN_WARNING "Failed to send QUIT "
-          "message to daemon [%d]; rc = [%d]\n",
+          "message to daemon [0x%p]; rc = [%d]\n",
           old_daemon->pid, rc);
   hlist_del(&old_daemon->euid_chain);
   kfree(old_daemon);
  }
- rc = ecryptfs_spawn_daemon(&new_daemon, euid, pid);
```

```
+ rc = ecryptfs_spawn_daemon(&new_daemon, euid, user_ns, pid);
  if (rc)
   printk(KERN_ERR "%s: The gods are displeased with this attempt "
-         "to create a new daemon object for euid [%d]; pid [%d]; "
-         "rc = [%d]\n", __func__, euid, pid, rc);
+          "to create a new daemon object for euid [%d]; pid "
+          "[0x%p]; rc = [%d]\n", __func__, euid, pid, rc);
  mutex_unlock(&ecryptfs_daemon_hash_mux);
  return rc;
 }
@@ -277,7 +284,7 @@ int ecryptfs_exorcise_daemon(struct ecryptfs_daemon *daemon)
      || (daemon->flags & ECRYPTFS_DAEMON_IN_POLL)) {
   rc = -EBUSY;
   printk(KERN_WARNING "%s: Attempt to destroy daemon with pid "
-         "[%d], but it is in the midst of a read or a poll\n",
+          "[0x%p], but it is in the midst of a read or a poll\n",
          __func__, daemon->pid);
   mutex_unlock(&daemon->mux);
   goto out;
@@ -303,6 +310,7 @@ out:
 /**
  * ecryptfs_process_quit
  * @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
  *       message
  *
@@ -310,17 +318,18 @@ out:
  * it is the registered that is requesting the deletion. Returns zero
  * after deleting the desired daemon; non-zero otherwise.
  */
-int ecryptfs_process_quit(uid_t euid, pid_t pid)
+int ecryptfs_process_quit(uid_t euid, struct user_namespace *user_ns,
+     struct pid *pid)
 {
  struct ecryptfs_daemon *daemon;
  int rc;

  mutex_lock(&ecryptfs_daemon_hash_mux);
- rc = ecryptfs_find_daemon_by_euid(&daemon, euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, euid, user_ns);
  if (rc || !daemon) {
   rc = -EINVAL;
   printk(KERN_ERR "Received request from user [%d] to "
-         "unregister unrecognized daemon [%d]\n", euid, pid);
+          "unregister unrecognized daemon [0x%p]\n", euid, pid);
   goto out_unlock;
  }
```

```
  rc = ecryptfs_exorcise_daemon(daemon);
@@ -354,11 +363,13 @@ out_unlock:
  * Returns zero on success; non-zero otherwise
  */
 int ecryptfs_process_response(struct ecryptfs_message *msg, uid_t euid,
-         pid_t pid, u32 seq)
+         struct user_namespace *user_ns, struct pid *pid,
+         u32 seq)
 {
  struct ecryptfs_daemon *daemon;
  struct ecryptfs_msg_ctx *msg_ctx;
  size_t msg_size;
+ struct nsproxy *nsproxy;
  int rc;

  if (msg->index >= ecryptfs_message_buf_len) {
@@ -372,12 +383,24 @@ int ecryptfs_process_response(struct ecryptfs_message *msg, uid_t
euid,
  msg_ctx = &ecryptfs_msg_ctx_arr[msg->index];
  mutex_lock(&msg_ctx->mux);
  mutex_lock(&ecryptfs_daemon_hash_mux);
- rc = ecryptfs_find_daemon_by_euid(&daemon, msg_ctx->task->euid);
+ rcu_read_lock();
+ nsproxy = task_nsproxy(msg_ctx->task);
+ if (nsproxy == NULL) {
+  rc = -EBADMSG;
+  printk(KERN_ERR "%s: Receiving process is a zombie. Dropping "
+         "message.\n", __func__);
+  rcu_read_unlock();
+  mutex_unlock(&ecryptfs_daemon_hash_mux);
+  goto wake_up;
+ }
+ rc = ecryptfs_find_daemon_by_euid(&daemon, msg_ctx->task->euid,
+       nsproxy->user_ns);
+ rcu_read_unlock();
  mutex_unlock(&ecryptfs_daemon_hash_mux);
  if (rc) {
   rc = -EBADMSG;
   printk(KERN_WARNING "%s: User [%d] received a "
-         "message response from process [%d] but does "
+         "message response from process [0x%p] but does "
          "not have a registered daemon\n", __func__,
          msg_ctx->task->euid, pid);
   goto wake_up;
@@ -389,10 +412,17 @@ int ecryptfs_process_response(struct ecryptfs_message *msg, uid_t
euid,
          euid, msg_ctx->task->euid);
   goto unlock;
```

```
  }
+ if (nsproxy->user_ns != user_ns) {
+  rc = -EBADMSG;
+  printk(KERN_WARNING "%s: Received message from user_ns "
+         "[0x%p]; expected message from user_ns [0x%p]\n",
+         __func__, user_ns, nsproxy->user_ns);
+  goto unlock;
+ }
  if (daemon->pid != pid) {
   rc = -EBADMSG;
   printk(KERN_ERR "%s: User [%d] sent a message response "
-         "from an unrecognized process [%d]\n",
+         "from an unrecognized process [0x%p]\n",
          __func__, msg_ctx->task->euid, pid);
   goto unlock;
  }
@@ -446,7 +476,8 @@ ecryptfs_send_message_locked(unsigned int transport, char *data, int
data_len,
  struct ecryptfs_daemon *daemon;
  int rc;

- rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid,
+        current->nsproxy->user_ns);
  if (rc || !daemon) {
   rc = -ENOTCONN;
   printk(KERN_ERR "%s: User [%d] does not have a daemon "
diff --git a/fs/ecryptfs/miscdev.c b/fs/ecryptfs/miscdev.c
index f7f2d90..8d39a04 100644
--- a/fs/ecryptfs/miscdev.c
+++ b/fs/ecryptfs/miscdev.c
@@ -46,7 +46,8 @@ ecryptfs_miscdev_poll(struct file *file, poll_table *pt)

  mutex_lock(&ecryptfs_daemon_hash_mux);
  /* TODO: Just use file->private_data? */
- rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid,
+        current->nsproxy->user_ns);
  BUG_ON(rc || !daemon);
  mutex_lock(&daemon->mux);
  mutex_unlock(&ecryptfs_daemon_hash_mux);
@@ -92,10 +93,12 @@ ecryptfs_miscdev_open(struct inode *inode, struct file *file)
         "count; rc = [%d]\n", __func__, rc);
   goto out_unlock_daemon_list;
  }
- rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid,
+        current->nsproxy->user_ns);
```

```
  if (rc || !daemon) {
   rc = ecryptfs_spawn_daemon(&daemon, current->euid,
-        current->pid);
+        current->nsproxy->user_ns,
+        task_pid(current));
   if (rc) {
    printk(KERN_ERR "%s: Error attempting to spawn daemon; "
           "rc = [%d]\n", __func__, rc);
@@ -103,18 +106,18 @@ ecryptfs_miscdev_open(struct inode *inode, struct file *file)
   }
  }
  mutex_lock(&daemon->mux);
- if (daemon->pid != current->pid) {
+ if (daemon->pid != task_pid(current)) {
   rc = -EINVAL;
-  printk(KERN_ERR "%s: pid [%d] has registered with euid [%d], "
-         "but pid [%d] has attempted to open the handle "
+  printk(KERN_ERR "%s: pid [0x%p] has registered with euid [%d], "
+         "but pid [0x%p] has attempted to open the handle "
         "instead\n", __func__, daemon->pid, daemon->euid,
-         current->pid);
+         task_pid(current));
   goto out_unlock_daemon;
  }
  if (daemon->flags & ECRYPTFS_DAEMON_MISCDEV_OPEN) {
   rc = -EBUSY;
   printk(KERN_ERR "%s: Miscellaneous device handle may only be "
-         "opened once per daemon; pid [%d] already has this "
+         "opened once per daemon; pid [0x%p] already has this "
         "handle open\n", __func__, daemon->pid);
   goto out_unlock_daemon;
  }
@@ -147,10 +150,11 @@ ecryptfs_miscdev_release(struct inode *inode, struct file *file)
  int rc;

  mutex_lock(&ecryptfs_daemon_hash_mux);
- rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid,
+       current->nsproxy->user_ns);
  BUG_ON(rc || !daemon);
  mutex_lock(&daemon->mux);
- BUG_ON(daemon->pid != current->pid);
+ BUG_ON(daemon->pid != task_pid(current));
  BUG_ON(!(daemon->flags & ECRYPTFS_DAEMON_MISCDEV_OPEN));
  daemon->flags &= ~ECRYPTFS_DAEMON_MISCDEV_OPEN;
  atomic_dec(&ecryptfs_num_miscdev_opens);
@@ -247,7 +251,8 @@ ecryptfs_miscdev_read(struct file *file, char __user *buf, size_t count,
```

```
  mutex_lock(&ecryptfs_daemon_hash_mux);
  /* TODO: Just use file->private_data? */
- rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid);
+ rc = ecryptfs_find_daemon_by_euid(&daemon, current->euid,
+        current->nsproxy->user_ns);
  BUG_ON(rc || !daemon);
  mutex_lock(&daemon->mux);
  if (daemon->flags & ECRYPTFS_DAEMON_ZOMBIE) {
@@ -285,7 +290,8 @@ check_list:
   goto check_list;
  }
  BUG_ON(current->euid != daemon->euid);
- BUG_ON(current->pid != daemon->pid);
+ BUG_ON(current->nsproxy->user_ns != daemon->user_ns);
+ BUG_ON(task_pid(current) != daemon->pid);
  msg_ctx = list_first_entry(&daemon->msg_ctx_out_queue,
        struct ecryptfs_msg_ctx, daemon_out_list);
  BUG_ON(!msg_ctx);
@@ -355,15 +361,18 @@ out_unlock_daemon:
 /**
  * ecryptfs_miscdev_helo
  * @euid: effective user id of miscdevess sending helo packet
+ * @user_ns: The namespace in which @euid applies
  * @pid: miscdevess id of miscdevess sending helo packet
  *
  * Returns zero on success; non-zero otherwise
  */
-static int ecryptfs_miscdev_helo(uid_t uid, pid_t pid)
+static int ecryptfs_miscdev_helo(uid_t euid, struct user_namespace *user_ns,
+     struct pid *pid)
 {
  int rc;

- rc = ecryptfs_process_helo(ECRYPTFS_TRANSPORT_MISCDEV, uid, pid);
+ rc = ecryptfs_process_helo(ECRYPTFS_TRANSPORT_MISCDEV, euid, user_ns,
+        pid);
  if (rc)
   printk(KERN_WARNING "Error processing HELO; rc = [%d]\n", rc);
  return rc;
@@ -372,15 +381,17 @@ static int ecryptfs_miscdev_helo(uid_t uid, pid_t pid)
 /**
  * ecryptfs_miscdev_quit
  * @euid: effective user id of miscdevess sending quit packet
+ * @user_ns: The namespace in which @euid applies
  * @pid: miscdevess id of miscdevess sending quit packet
  *
  * Returns zero on success; non-zero otherwise
  */
```

```
-static int ecryptfs_miscdev_quit(uid_t euid, pid_t pid)
+static int ecryptfs_miscdev_quit(uid_t euid, struct user_namespace *user_ns,
+     struct pid *pid)
 {
  int rc;

- rc = ecryptfs_process_quit(euid, pid);
+ rc = ecryptfs_process_quit(euid, user_ns, pid);
  if (rc)
   printk(KERN_WARNING
          "Error processing QUIT message; rc = [%d]\n", rc);
@@ -392,13 +403,15 @@ static int ecryptfs_miscdev_quit(uid_t euid, pid_t pid)
  * @data: Bytes comprising struct ecryptfs_message
  * @data_size: sizeof(struct ecryptfs_message) + data len
  * @euid: Effective user id of miscdevess sending the miscdev response
+ * @user_ns: The namespace in which @euid applies
  * @pid: Miscdevess id of miscdevess sending the miscdev response
  * @seq: Sequence number for miscdev response packet
  *
  * Returns zero on success; non-zero otherwise
  */
 static int ecryptfs_miscdev_response(char *data, size_t data_size,
-       uid_t euid, pid_t pid, u32 seq)
+         uid_t euid, struct user_namespace *user_ns,
+         struct pid *pid, u32 seq)
 {
  struct ecryptfs_message *msg = (struct ecryptfs_message *)data;
  int rc;
@@ -410,7 +423,7 @@ static int ecryptfs_miscdev_response(char *data, size_t data_size,
   rc = -EINVAL;
   goto out;
  }
- rc = ecryptfs_process_response(msg, euid, pid, seq);
+ rc = ecryptfs_process_response(msg, euid, user_ns, pid, seq);
  if (rc)
   printk(KERN_ERR
          "Error processing response message; rc = [%d]\n", rc);
@@ -491,27 +504,32 @@ ecryptfs_miscdev_write(struct file *file, const char __user *buf,
   }
   rc = ecryptfs_miscdev_response(&data[i], packet_size,
            current->euid,
-           current->pid, seq);
+           current->nsproxy->user_ns,
+           task_pid(current), seq);
   if (rc)
    printk(KERN_WARNING "%s: Failed to deliver miscdev "
          "response to requesting operation; rc = [%d]\n",
          __func__, rc);
```

```
   break;
  case ECRYPTFS_MSG_HELO:
-  rc = ecryptfs_miscdev_helo(current->euid, current->pid);
+  rc = ecryptfs_miscdev_helo(current->euid,
+         current->nsproxy->user_ns,
+         task_pid(current));
   if (rc) {
    printk(KERN_ERR "%s: Error attempting to process "
-          "helo from pid [%d]; rc = [%d]\n", __func__,
-          current->pid, rc);
+          "helo from pid [0x%p]; rc = [%d]\n", __func__,
+          task_pid(current), rc);
    goto out_free;
   }
   break;
  case ECRYPTFS_MSG_QUIT:
-  rc = ecryptfs_miscdev_quit(current->euid, current->pid);
+  rc = ecryptfs_miscdev_quit(current->euid,
+         current->nsproxy->user_ns,
+         task_pid(current));
   if (rc) {
    printk(KERN_ERR "%s: Error attempting to process "
-          "quit from pid [%d]; rc = [%d]\n", __func__,
-          current->pid, rc);
+          "quit from pid [0x%p]; rc = [%d]\n", __func__,
+          task_pid(current), rc);
    goto out_free;
   }
   break;
diff --git a/fs/ecryptfs/netlink.c b/fs/ecryptfs/netlink.c
index eb70f69..e0abad6 100644
--- a/fs/ecryptfs/netlink.c
+++ b/fs/ecryptfs/netlink.c
@@ -45,7 +45,7 @@ static struct sock *ecryptfs_nl_sock;
  */
 int ecryptfs_send_netlink(char *data, int data_len,
      struct ecryptfs_msg_ctx *msg_ctx, u8 msg_type,
-     u16 msg_flags, pid_t daemon_pid)
+     u16 msg_flags, struct pid *daemon_pid)
 {
  struct sk_buff *skb;
  struct nlmsghdr *nlh;
@@ -60,7 +60,7 @@ int ecryptfs_send_netlink(char *data, int data_len,
   ecryptfs_printk(KERN_ERR, "Failed to allocate socket buffer\n");
   goto out;
  }
- nlh = NLMSG_PUT(skb, daemon_pid, msg_ctx ? msg_ctx->counter : 0,
+ 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) {
@@ -69,7 +69,7 @@ int ecryptfs_send_netlink(char *data, int data_len,
    msg->data_len = data_len;
    memcpy(msg->data, data, data_len);
  }
- rc = netlink_unicast(ecryptfs_nl_sock, skb, daemon_pid, 0);
+ 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);
@@ -99,6 +99,7 @@ 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) {
@@ -107,8 +108,10 @@ static int ecryptfs_process_nl_response(struct sk_buff *skb)
     "incorrectly specified data length\n");
    goto out;
  }
- rc = ecryptfs_process_response(msg, NETLINK_CREDS(skb)->uid,
-          NETLINK_CREDS(skb)->pid, nlh->nlmsg_seq);
+ 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);
@@ -126,11 +129,13 @@ out:
  */
 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,
-       NETLINK_CREDS(skb)->pid);
+       NETLINK_CREDS(skb)->uid, NULL, pid);
+ put_pid(pid);
  if (rc)
    printk(KERN_WARNING "Error processing HELO; rc = [%d]\n", rc);
```

```
  return rc;
@@ -147,10 +152,12 @@ static int ecryptfs_process_nl_helo(struct sk_buff *skb)
  */
 static int ecryptfs_process_nl_quit(struct sk_buff *skb)
 {
+ struct pid *pid;
  int rc;

- rc = ecryptfs_process_quit(NETLINK_CREDS(skb)->uid,
-       NETLINK_CREDS(skb)->pid);
+ 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);
--
1.5.3.6
```

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
```

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