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
```