Subject: [patch i2o] i2o layer cleanup
Posted by vaverin on Tue, 15 May 2007 12:41:16 GMT
View Forum Message <> Reply to Message

Andrew,

I've fixed a number of issues in i2o layer:
[patch i2o 1/6] i2o_cfg_passthru cleanup (memory leak and infinite loop)
[patch i2o 2/6] wrong memory access in i2o_block_device_lock()
[patch i2o 3/6] i2o message leak in i2o_msg_post_wait_mem()
[patch i2o 4/6] i2o proc reading oops
[patch i2o 5/6] i2o_proc files permission
[patch i2o 6/6] i2o debug output cleanup

However because of Markus Lidel is not i2o maintainer now, I do not understand
who should agree the following patches.

Thank you,
 Vasily Averin

Subject: [patch i2o 1/6] i2o_cfg_passthru cleanup
Posted by vaverin on Tue, 15 May 2007 12:42:26 GMT
View Forum Message <> Reply to Message

This patch fixes a number of issues in i2o_cfg_passthru{,32}:
- i2o_msg_get_wait() return vaile is not checked;
- i2o_message memory leaks on error paths;
- infinite loop to sg_list_cleanup in passthru32

it's important issue because of i2o_cfg_passthru is used by raidutils for
monitorig controllers state, and in case of memory shortage it leads to the
node crash or disk IO stall.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/i2o_config.c
+++ lk2.6/drivers/message/i2o/i2o_config.c
@@ -554,8 +554,6 @@ static int i2o_cfg_passthru32(struct fil
   return -ENXIO;
  }

- msg = i2o_msg_get_wait(c, I2O_TIMEOUT_MESSAGE_GET);
-
  sb = c->status_block.virt;

  if (get_user(size, &user_msg[0])) {
@@ -573,24 +571,30 @@ static int i2o_cfg_passthru32(struct fil
```

```
  size <<= 2;  // Convert to bytes

+ msg = i2o_msg_get_wait(c, I2O_TIMEOUT_MESSAGE_GET);
+ if (IS_ERR(msg))
+  return PTR_ERR(msg);
+
+ rcode = -EFAULT;
  /* Copy in the user's I2O command */
  if (copy_from_user(msg, user_msg, size)) {
   osm_warn("unable to copy user message\n");
-  return -EFAULT;
+  goto out;
  }
  i2o_dump_message(msg);

  if (get_user(reply_size, &user_reply[0]) < 0)
-  return -EFAULT;
+  goto out;

  reply_size >>= 16;
  reply_size <<= 2;

+ rcode = -ENOMEM;
  reply = kzalloc(reply_size, GFP_KERNEL);
  if (!reply) {
   printk(KERN_WARNING "%s: Could not allocate reply buffer\n",
          c->name);
-  return -ENOMEM;
+  goto out;
  }

  sg_offset = (msg->u.head[0] >> 4) & 0x0f;
@@ -661,13 +665,14 @@ static int i2o_cfg_passthru32(struct fil
  }

  rcode = i2o_msg_post_wait(c, msg, 60);
+ msg = NULL;
  if (rcode) {
   reply[4] = ((u32) rcode) << 24;
   goto sg_list_cleanup;
  }

  if (sg_offset) {
-  u32 msg[I2O_OUTBOUND_MSG_FRAME_SIZE];
+  u32 rmsg[I2O_OUTBOUND_MSG_FRAME_SIZE];
   /* Copy back the Scatter Gather buffers back to user space */
   u32 j;
```

```
   // TODO 64bit fix
@@ -675,7 +680,7 @@ static int i2o_cfg_passthru32(struct fil
   int sg_size;

   // re-acquire the original message to handle correctly the sg copy operation
-  memset(&msg, 0, I2O_OUTBOUND_MSG_FRAME_SIZE * 4);
+  memset(&rmsg, 0, I2O_OUTBOUND_MSG_FRAME_SIZE * 4);
   // get user msg size in u32s
   if (get_user(size, &user_msg[0])) {
    rcode = -EFAULT;
@@ -684,7 +689,7 @@ static int i2o_cfg_passthru32(struct fil
   size = size >> 16;
   size *= 4;
   /* Copy in the user's I2O command */
-  if (copy_from_user(msg, user_msg, size)) {
+  if (copy_from_user(rmsg, user_msg, size)) {
    rcode = -EFAULT;
    goto sg_list_cleanup;
   }
@@ -692,7 +697,7 @@ static int i2o_cfg_passthru32(struct fil
       (size - sg_offset * 4) / sizeof(struct sg_simple_element);

   // TODO 64bit fix
-  sg = (struct sg_simple_element *)(msg + sg_offset);
+  sg = (struct sg_simple_element *)(rmsg + sg_offset);
   for (j = 0; j < sg_count; j++) {
    /* Copy out the SG list to user's buffer if necessary */
    if (!
@@ -714,7 +719,7 @@ static int i2o_cfg_passthru32(struct fil
   }
  }

-      sg_list_cleanup:
+sg_list_cleanup:
  /* Copy back the reply to user space */
  if (reply_size) {
   // we wrote our own values for context - now restore the user supplied ones
@@ -723,7 +728,6 @@ static int i2o_cfg_passthru32(struct fil
          "%s: Could not copy message context FROM user\n",
          c->name);
    rcode = -EFAULT;
-   goto sg_list_cleanup;
   }
   if (copy_to_user(user_reply, reply, reply_size)) {
    printk(KERN_WARNING
@@ -731,12 +735,14 @@ static int i2o_cfg_passthru32(struct fil
    rcode = -EFAULT;
   }
```

```
  }
-
  for (i = 0; i < sg_index; i++)
   i2o_dma_free(&c->pdev->dev, &sg_list[i]);

-      cleanup:
+cleanup:
  kfree(reply);
+ if (msg)
+out:
+  i2o_msg_nop(c, msg);
  return rcode;
 }

@@ -793,8 +799,6 @@ static int i2o_cfg_passthru(unsigned lon
   return -ENXIO;
  }

- msg = i2o_msg_get_wait(c, I2O_TIMEOUT_MESSAGE_GET);
-
  sb = c->status_block.virt;

  if (get_user(size, &user_msg[0]))
@@ -810,12 +814,17 @@ static int i2o_cfg_passthru(unsigned lon

  size <<= 2;  // Convert to bytes

+ msg = i2o_msg_get_wait(c, I2O_TIMEOUT_MESSAGE_GET);
+ if (IS_ERR(msg))
+  return PTR_ERR(msg);
+
+ rcode = -EFAULT;
  /* Copy in the user's I2O command */
  if (copy_from_user(msg, user_msg, size))
-  return -EFAULT;
+  goto out;

  if (get_user(reply_size, &user_reply[0]) < 0)
-  return -EFAULT;
+  goto out;

  reply_size >>= 16;
  reply_size <<= 2;
@@ -824,7 +833,8 @@ static int i2o_cfg_passthru(unsigned lon
  if (!reply) {
   printk(KERN_WARNING "%s: Could not allocate reply buffer\n",
          c->name);
-  return -ENOMEM;
```

```
+  rcode = -ENOMEM;
+  goto out;
  }

  sg_offset = (msg->u.head[0] >> 4) & 0x0f;
@@ -891,13 +901,14 @@ static int i2o_cfg_passthru(unsigned lon
  }

  rcode = i2o_msg_post_wait(c, msg, 60);
+ msg = NULL;
  if (rcode) {
   reply[4] = ((u32) rcode) << 24;
   goto sg_list_cleanup;
  }

  if (sg_offset) {
-  u32 msg[128];
+  u32 rmsg[128];
   /* Copy back the Scatter Gather buffers back to user space */
   u32 j;
   // TODO 64bit fix
@@ -905,7 +916,7 @@ static int i2o_cfg_passthru(unsigned lon
   int sg_size;

   // re-acquire the original message to handle correctly the sg copy operation
-  memset(&msg, 0, I2O_OUTBOUND_MSG_FRAME_SIZE * 4);
+  memset(&rmsg, 0, I2O_OUTBOUND_MSG_FRAME_SIZE * 4);
   // get user msg size in u32s
   if (get_user(size, &user_msg[0])) {
    rcode = -EFAULT;
@@ -914,7 +925,7 @@ static int i2o_cfg_passthru(unsigned lon
   size = size >> 16;
   size *= 4;
   /* Copy in the user's I2O command */
-  if (copy_from_user(msg, user_msg, size)) {
+  if (copy_from_user(rmsg, user_msg, size)) {
    rcode = -EFAULT;
    goto sg_list_cleanup;
   }
@@ -922,7 +933,7 @@ static int i2o_cfg_passthru(unsigned lon
       (size - sg_offset * 4) / sizeof(struct sg_simple_element);

   // TODO 64bit fix
-  sg = (struct sg_simple_element *)(msg + sg_offset);
+  sg = (struct sg_simple_element *)(rmsg + sg_offset);
   for (j = 0; j < sg_count; j++) {
    /* Copy out the SG list to user's buffer if necessary */
    if (!
```

```
@@ -944,7 +955,7 @@ static int i2o_cfg_passthru(unsigned lon
   }
  }

-      sg_list_cleanup:
+sg_list_cleanup:
  /* Copy back the reply to user space */
  if (reply_size) {
   // we wrote our own values for context - now restore the user supplied ones
@@ -964,8 +975,11 @@ static int i2o_cfg_passthru(unsigned lon
  for (i = 0; i < sg_index; i++)
   kfree(sg_list[i]);

-      cleanup:
+cleanup:
  kfree(reply);
+ if (msg)
+out:
+  i2o_msg_nop(c, msg);
  return rcode;
 }
 #endif
```

Subject: [patch i2o 2/6] wrong memory access in i2o_block_device_lock()
Posted by vaverin on Tue, 15 May 2007 12:43:17 GMT
View Forum Message <> Reply to Message

This patch fixes access to memory that has not been allocated:
i2o_msg_get_wait() can returns errors different from I2O_QUEUE_EMPTY. But the
result is checked only against this code. If it is not I2O_QUEUE_EMPTY then we
dereference the error code as the pointer later.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/i2o_block.c
+++ lk2.6/drivers/message/i2o/i2o_block.c
@@ -215,7 +215,7 @@ static int i2o_block_device_lock(struct
  struct i2o_message *msg;

  msg = i2o_msg_get_wait(dev->iop, I2O_TIMEOUT_MESSAGE_GET);
- if (IS_ERR(msg) == I2O_QUEUE_EMPTY)
+ if (IS_ERR(msg))
   return PTR_ERR(msg);

  msg->u.head[0] = cpu_to_le32(FIVE_WORD_MSG_SIZE | SGL_OFFSET_0);
```

Subject: [patch i2o 3/6] i2o message leak in i2o_msg_post_wait_mem()
Posted by vaverin on Tue, 15 May 2007 12:44:27 GMT
View Forum Message <> Reply to Message

We need to free i2o msg in case of error.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/exec-osm.c
+++ lk2.6/drivers/message/i2o/exec-osm.c
@@ -131,8 +131,10 @@ int i2o_msg_post_wait_mem(struct i2o_con
  int rc = 0;

  wait = i2o_exec_wait_alloc();
- if (!wait)
+ if (!wait) {
+  i2o_msg_nop(c, msg);
   return -ENOMEM;
+ }

  if (tcntxt == 0xffffffff)
   tcntxt = 0x80000000;
```

Subject: [patch i2o 4/6] i2o proc reading oops
Posted by vaverin on Tue, 15 May 2007 12:45:20 GMT
View Forum Message <> Reply to Message

fixed oops on reading from some i2o proc files (i2o_seq_show_driver_store() and
other) because their handlers uses "exec" field in struct i2o_controller

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/exec-osm.c
+++ lk2.6/drivers/message/i2o/exec-osm.c
@@ -339,6 +339,8 @@ static int i2o_exec_probe(struct device
  rc = device_create_file(dev, &dev_attr_product_id);
  if (rc) goto err_vid;

+ i2o_dev->iop->exec = i2o_dev;
+
  return 0;

 err_vid:
```

Subject: [patch i2o 5/6] i2o_proc files permission

Posted by vaverin on Tue, 15 May 2007 12:47:05 GMT
View Forum Message <> Reply to Message

Reading from some i2o related proc files can lead to the i2o controller hang due
unknown reasons. As a workaround this patch changes the permission of these
files to root-only accessible.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/i2o_proc.c
+++ lk2.6/drivers/message/i2o/i2o_proc.c
@@ -1855,17 +1855,17 @@ static i2o_proc_entry i2o_proc_generic_i
  * Device specific entries
  */
 static i2o_proc_entry generic_dev_entries[] = {
- {"groups", S_IFREG | S_IRUGO, &i2o_seq_fops_groups},
- {"phys_dev", S_IFREG | S_IRUGO, &i2o_seq_fops_phys_device},
- {"claimed", S_IFREG | S_IRUGO, &i2o_seq_fops_claimed},
- {"users", S_IFREG | S_IRUGO, &i2o_seq_fops_users},
- {"priv_msgs", S_IFREG | S_IRUGO, &i2o_seq_fops_priv_msgs},
- {"authorized_users", S_IFREG | S_IRUGO, &i2o_seq_fops_authorized_users},
- {"dev_identity", S_IFREG | S_IRUGO, &i2o_seq_fops_dev_identity},
- {"ddm_identity", S_IFREG | S_IRUGO, &i2o_seq_fops_ddm_identity},
- {"user_info", S_IFREG | S_IRUGO, &i2o_seq_fops_uinfo},
- {"sgl_limits", S_IFREG | S_IRUGO, &i2o_seq_fops_sgl_limits},
- {"sensors", S_IFREG | S_IRUGO, &i2o_seq_fops_sensors},
+ {"groups", S_IFREG | S_IRUSR, &i2o_seq_fops_groups},
+ {"phys_dev", S_IFREG | S_IRUSR, &i2o_seq_fops_phys_device},
+ {"claimed", S_IFREG | S_IRUSR, &i2o_seq_fops_claimed},
+ {"users", S_IFREG | S_IRUSR, &i2o_seq_fops_users},
+ {"priv_msgs", S_IFREG | S_IRUSR, &i2o_seq_fops_priv_msgs},
+ {"authorized_users", S_IFREG | S_IRUSR, &i2o_seq_fops_authorized_users},
+ {"dev_identity", S_IFREG | S_IRUSR, &i2o_seq_fops_dev_identity},
+ {"ddm_identity", S_IFREG | S_IRUSR, &i2o_seq_fops_ddm_identity},
+ {"user_info", S_IFREG | S_IRUSR, &i2o_seq_fops_uinfo},
+ {"sgl_limits", S_IFREG | S_IRUSR, &i2o_seq_fops_sgl_limits},
+ {"sensors", S_IFREG | S_IRUSR, &i2o_seq_fops_sensors},
   {NULL, 0, NULL}
 };
```

Subject: [patch i2o 6/6] i2o debug output cleanup
Posted by vaverin on Tue, 15 May 2007 12:48:16 GMT
View Forum Message <> Reply to Message

fixed output of i2o debug messages, extra KERN_ are removed

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- lk2.6/drivers/message/i2o/debug.c
+++ lk2.6/drivers/message/i2o/debug.c
@@ -24,7 +24,7 @@ void i2o_report_status(const char *sever
  if (cmd == I2O_CMD_UTIL_EVT_REGISTER)
   return;  // No status in this reply

- printk(KERN_DEBUG "%s%s: ", severity, str);
+ printk("%s%s: ", severity, str);

  if (cmd < 0x1F)  // Utility cmd
   i2o_report_util_cmd(cmd);
@@ -32,7 +32,7 @@ void i2o_report_status(const char *sever
  else if (cmd >= 0xA0 && cmd <= 0xEF) // Executive cmd
   i2o_report_exec_cmd(cmd);
  else
-  printk(KERN_DEBUG "Cmd = %0#2x, ", cmd); // Other cmds
+  printk("Cmd = %0#2x, ", cmd); // Other cmds

  if (msg[0] & MSG_FAIL) {
   i2o_report_fail_status(req_status, msg);
@@ -44,7 +44,7 @@ void i2o_report_status(const char *sever
  if (cmd < 0x1F || (cmd >= 0xA0 && cmd <= 0xEF))
   i2o_report_common_dsc(detailed_status);
  else
-  printk(KERN_DEBUG " / DetailedStatus = %0#4x.\n",
+  printk(" / DetailedStatus = %0#4x.\n",
         detailed_status);
 }

@@ -89,10 +89,10 @@ static void i2o_report_fail_status(u8 re
  };

  if (req_status == I2O_FSC_TRANSPORT_UNKNOWN_FAILURE)
-  printk(KERN_DEBUG "TRANSPORT_UNKNOWN_FAILURE (%0#2x).\n",
+  printk("TRANSPORT_UNKNOWN_FAILURE (%0#2x).\n",
         req_status);
  else
-  printk(KERN_DEBUG "TRANSPORT_%s.\n",
+  printk("TRANSPORT_%s.\n",
         FAIL_STATUS[req_status & 0x0F]);

  /* Dump some details */
@@ -104,7 +104,7 @@ static void i2o_report_fail_status(u8 re
  printk(KERN_ERR "  FailingHostUnit = 0x%04X,  FailingIOP = 0x%03X\n",
        msg[5] >> 16, msg[5] & 0xFFF);

- printk(KERN_ERR "  Severity:  0x%02X ", (msg[4] >> 16) & 0xFF);
```

```
+ printk(KERN_ERR "  Severity:  0x%02X\n", (msg[4] >> 16) & 0xFF);
  if (msg[4] & (1 << 16))
   printk(KERN_DEBUG "(FormatError), "
          "this msg can never be delivered/processed.\n");
@@ -142,9 +142,9 @@ static void i2o_report_common_status(u8
  };

  if (req_status >= ARRAY_SIZE(REPLY_STATUS))
-  printk(KERN_DEBUG "RequestStatus = %0#2x", req_status);
+  printk("RequestStatus = %0#2x", req_status);
  else
-  printk(KERN_DEBUG "%s", REPLY_STATUS[req_status]);
+  printk("%s", REPLY_STATUS[req_status]);
 }

 /*
@@ -187,10 +187,10 @@ static void i2o_report_common_dsc(u16 de
  };

  if (detailed_status > I2O_DSC_DEVICE_NOT_AVAILABLE)
-  printk(KERN_DEBUG " / DetailedStatus = %0#4x.\n",
+  printk(" / DetailedStatus = %0#4x.\n",
          detailed_status);
  else
-  printk(KERN_DEBUG " / %s.\n", COMMON_DSC[detailed_status]);
+  printk(" / %s.\n", COMMON_DSC[detailed_status]);
 }

 /*
@@ -200,49 +200,49 @@ static void i2o_report_util_cmd(u8 cmd)
 {
  switch (cmd) {
  case I2O_CMD_UTIL_NOP:
-  printk(KERN_DEBUG "UTIL_NOP, ");
+  printk("UTIL_NOP, ");
   break;
  case I2O_CMD_UTIL_ABORT:
-  printk(KERN_DEBUG "UTIL_ABORT, ");
+  printk("UTIL_ABORT, ");
   break;
  case I2O_CMD_UTIL_CLAIM:
-  printk(KERN_DEBUG "UTIL_CLAIM, ");
+  printk("UTIL_CLAIM, ");
   break;
  case I2O_CMD_UTIL_RELEASE:
-  printk(KERN_DEBUG "UTIL_CLAIM_RELEASE, ");
+  printk("UTIL_CLAIM_RELEASE, ");
   break;
```

```
  case I2O_CMD_UTIL_CONFIG_DIALOG:
-  printk(KERN_DEBUG "UTIL_CONFIG_DIALOG, ");
+  printk("UTIL_CONFIG_DIALOG, ");
    break;
  case I2O_CMD_UTIL_DEVICE_RESERVE:
-  printk(KERN_DEBUG "UTIL_DEVICE_RESERVE, ");
+  printk("UTIL_DEVICE_RESERVE, ");
    break;
  case I2O_CMD_UTIL_DEVICE_RELEASE:
-  printk(KERN_DEBUG "UTIL_DEVICE_RELEASE, ");
+  printk("UTIL_DEVICE_RELEASE, ");
    break;
  case I2O_CMD_UTIL_EVT_ACK:
-  printk(KERN_DEBUG "UTIL_EVENT_ACKNOWLEDGE, ");
+  printk("UTIL_EVENT_ACKNOWLEDGE, ");
    break;
  case I2O_CMD_UTIL_EVT_REGISTER:
-  printk(KERN_DEBUG "UTIL_EVENT_REGISTER, ");
+  printk("UTIL_EVENT_REGISTER, ");
    break;
  case I2O_CMD_UTIL_LOCK:
-  printk(KERN_DEBUG "UTIL_LOCK, ");
+  printk("UTIL_LOCK, ");
    break;
  case I2O_CMD_UTIL_LOCK_RELEASE:
-  printk(KERN_DEBUG "UTIL_LOCK_RELEASE, ");
+  printk("UTIL_LOCK_RELEASE, ");
    break;
  case I2O_CMD_UTIL_PARAMS_GET:
-  printk(KERN_DEBUG "UTIL_PARAMS_GET, ");
+  printk("UTIL_PARAMS_GET, ");
    break;
  case I2O_CMD_UTIL_PARAMS_SET:
-  printk(KERN_DEBUG "UTIL_PARAMS_SET, ");
+  printk("UTIL_PARAMS_SET, ");
    break;
  case I2O_CMD_UTIL_REPLY_FAULT_NOTIFY:
-  printk(KERN_DEBUG "UTIL_REPLY_FAULT_NOTIFY, ");
+  printk("UTIL_REPLY_FAULT_NOTIFY, ");
    break;
  default:
-  printk(KERN_DEBUG "Cmd = %0#2x, ", cmd);
+  printk("Cmd = %0#2x, ", cmd);
  }
 }

@@ -253,106 +253,106 @@ static void i2o_report_exec_cmd(u8 cmd)
 {
```

```
  switch (cmd) {
  case I2O_CMD_ADAPTER_ASSIGN:
-  printk(KERN_DEBUG "EXEC_ADAPTER_ASSIGN, ");
+  printk("EXEC_ADAPTER_ASSIGN, ");
   break;
  case I2O_CMD_ADAPTER_READ:
-  printk(KERN_DEBUG "EXEC_ADAPTER_READ, ");
+  printk("EXEC_ADAPTER_READ, ");
   break;
  case I2O_CMD_ADAPTER_RELEASE:
-  printk(KERN_DEBUG "EXEC_ADAPTER_RELEASE, ");
+  printk("EXEC_ADAPTER_RELEASE, ");
   break;
  case I2O_CMD_BIOS_INFO_SET:
-  printk(KERN_DEBUG "EXEC_BIOS_INFO_SET, ");
+  printk("EXEC_BIOS_INFO_SET, ");
   break;
  case I2O_CMD_BOOT_DEVICE_SET:
-  printk(KERN_DEBUG "EXEC_BOOT_DEVICE_SET, ");
+  printk("EXEC_BOOT_DEVICE_SET, ");
   break;
  case I2O_CMD_CONFIG_VALIDATE:
-  printk(KERN_DEBUG "EXEC_CONFIG_VALIDATE, ");
+  printk("EXEC_CONFIG_VALIDATE, ");
   break;
  case I2O_CMD_CONN_SETUP:
-  printk(KERN_DEBUG "EXEC_CONN_SETUP, ");
+  printk("EXEC_CONN_SETUP, ");
   break;
  case I2O_CMD_DDM_DESTROY:
-  printk(KERN_DEBUG "EXEC_DDM_DESTROY, ");
+  printk("EXEC_DDM_DESTROY, ");
   break;
  case I2O_CMD_DDM_ENABLE:
-  printk(KERN_DEBUG "EXEC_DDM_ENABLE, ");
+  printk("EXEC_DDM_ENABLE, ");
   break;
  case I2O_CMD_DDM_QUIESCE:
-  printk(KERN_DEBUG "EXEC_DDM_QUIESCE, ");
+  printk("EXEC_DDM_QUIESCE, ");
   break;
  case I2O_CMD_DDM_RESET:
-  printk(KERN_DEBUG "EXEC_DDM_RESET, ");
+  printk("EXEC_DDM_RESET, ");
   break;
  case I2O_CMD_DDM_SUSPEND:
-  printk(KERN_DEBUG "EXEC_DDM_SUSPEND, ");
+  printk("EXEC_DDM_SUSPEND, ");
```

```
   break;
  case I2O_CMD_DEVICE_ASSIGN:
-  printk(KERN_DEBUG "EXEC_DEVICE_ASSIGN, ");
+  printk("EXEC_DEVICE_ASSIGN, ");
   break;
  case I2O_CMD_DEVICE_RELEASE:
-  printk(KERN_DEBUG "EXEC_DEVICE_RELEASE, ");
+  printk("EXEC_DEVICE_RELEASE, ");
   break;
  case I2O_CMD_HRT_GET:
-  printk(KERN_DEBUG "EXEC_HRT_GET, ");
+  printk("EXEC_HRT_GET, ");
   break;
  case I2O_CMD_ADAPTER_CLEAR:
-  printk(KERN_DEBUG "EXEC_IOP_CLEAR, ");
+  printk("EXEC_IOP_CLEAR, ");
   break;
  case I2O_CMD_ADAPTER_CONNECT:
-  printk(KERN_DEBUG "EXEC_IOP_CONNECT, ");
+  printk("EXEC_IOP_CONNECT, ");
   break;
  case I2O_CMD_ADAPTER_RESET:
-  printk(KERN_DEBUG "EXEC_IOP_RESET, ");
+  printk("EXEC_IOP_RESET, ");
   break;
  case I2O_CMD_LCT_NOTIFY:
-  printk(KERN_DEBUG "EXEC_LCT_NOTIFY, ");
+  printk("EXEC_LCT_NOTIFY, ");
   break;
  case I2O_CMD_OUTBOUND_INIT:
-  printk(KERN_DEBUG "EXEC_OUTBOUND_INIT, ");
+  printk("EXEC_OUTBOUND_INIT, ");
   break;
  case I2O_CMD_PATH_ENABLE:
-  printk(KERN_DEBUG "EXEC_PATH_ENABLE, ");
+  printk("EXEC_PATH_ENABLE, ");
   break;
  case I2O_CMD_PATH_QUIESCE:
-  printk(KERN_DEBUG "EXEC_PATH_QUIESCE, ");
+  printk("EXEC_PATH_QUIESCE, ");
   break;
  case I2O_CMD_PATH_RESET:
-  printk(KERN_DEBUG "EXEC_PATH_RESET, ");
+  printk("EXEC_PATH_RESET, ");
   break;
  case I2O_CMD_STATIC_MF_CREATE:
-  printk(KERN_DEBUG "EXEC_STATIC_MF_CREATE, ");
+  printk("EXEC_STATIC_MF_CREATE, ");
```

```
   break;
  case I2O_CMD_STATIC_MF_RELEASE:
-  printk(KERN_DEBUG "EXEC_STATIC_MF_RELEASE, ");
+  printk("EXEC_STATIC_MF_RELEASE, ");
   break;
  case I2O_CMD_STATUS_GET:
-  printk(KERN_DEBUG "EXEC_STATUS_GET, ");
+  printk("EXEC_STATUS_GET, ");
   break;
  case I2O_CMD_SW_DOWNLOAD:
-  printk(KERN_DEBUG "EXEC_SW_DOWNLOAD, ");
+  printk("EXEC_SW_DOWNLOAD, ");
   break;
  case I2O_CMD_SW_UPLOAD:
-  printk(KERN_DEBUG "EXEC_SW_UPLOAD, ");
+  printk("EXEC_SW_UPLOAD, ");
   break;
  case I2O_CMD_SW_REMOVE:
-  printk(KERN_DEBUG "EXEC_SW_REMOVE, ");
+  printk("EXEC_SW_REMOVE, ");
   break;
  case I2O_CMD_SYS_ENABLE:
-  printk(KERN_DEBUG "EXEC_SYS_ENABLE, ");
+  printk("EXEC_SYS_ENABLE, ");
   break;
  case I2O_CMD_SYS_MODIFY:
-  printk(KERN_DEBUG "EXEC_SYS_MODIFY, ");
+  printk("EXEC_SYS_MODIFY, ");
   break;
  case I2O_CMD_SYS_QUIESCE:
-  printk(KERN_DEBUG "EXEC_SYS_QUIESCE, ");
+  printk("EXEC_SYS_QUIESCE, ");
   break;
  case I2O_CMD_SYS_TAB_SET:
-  printk(KERN_DEBUG "EXEC_SYS_TAB_SET, ");
+  printk("EXEC_SYS_TAB_SET, ");
   break;
  default:
-  printk(KERN_DEBUG "Cmd = %#02x, ", cmd);
+  printk("Cmd = %#02x, ", cmd);
  }
 }

@@ -361,28 +361,28 @@ void i2o_debug_state(struct i2o_controll
  printk(KERN_INFO "%s: State = ", c->name);
  switch (((i2o_status_block *) c->status_block.virt)->iop_state) {
  case 0x01:
-  printk(KERN_DEBUG "INIT\n");
```

```
+  printk("INIT\n");
   break;
  case 0x02:
-  printk(KERN_DEBUG "RESET\n");
+  printk("RESET\n");
   break;
  case 0x04:
-  printk(KERN_DEBUG "HOLD\n");
+  printk("HOLD\n");
   break;
  case 0x05:
-  printk(KERN_DEBUG "READY\n");
+  printk("READY\n");
   break;
  case 0x08:
-  printk(KERN_DEBUG "OPERATIONAL\n");
+  printk("OPERATIONAL\n");
   break;
  case 0x10:
-  printk(KERN_DEBUG "FAILED\n");
+  printk("FAILED\n");
   break;
  case 0x11:
-  printk(KERN_DEBUG "FAULTED\n");
+  printk("FAULTED\n");
   break;
  default:
-  printk(KERN_DEBUG "%x (unknown !!)\n",
+  printk("%x (unknown !!)\n",
         ((i2o_status_block *) c->status_block.virt)->iop_state);
 }
};
```

Subject: Re: [patch i2o] i2o layer cleanup
Posted by Kirill Korotaev on Tue, 15 May 2007 12:53:20 GMT
View Forum Message <> Reply to Message

Acked-By: Kirill Korotaev <dev@openvz.org>

Vasily Averin wrote:
> Andrew,
>
> I've fixed a number of issues in i2o layer:
> [patch i2o 1/6] i2o_cfg_passthru cleanup (memory leak and infinite loop)
> [patch i2o 2/6] wrong memory access in i2o_block_device_lock()
> [patch i2o 3/6] i2o message leak in i2o_msg_post_wait_mem()

> [patch i2o 4/6] i2o proc reading oops
> [patch i2o 5/6] i2o_proc files permission
> [patch i2o 6/6] i2o debug output cleanup
>
> However because of Markus Lidel is not i2o maintainer now, I do not understand
> who should agree the following patches.
>
> Thank you,
> Vasily Averin
>

Subject: Re: [patch i2o 5/6] i2o_proc files permission
Posted by vaverin on Tue, 15 May 2007 12:59:49 GMT
View Forum Message <> Reply to Message

I would add:
I've reported about this issue some time ago to security@kernel.org
How this lockup can be reproduced:
- boot the kernel,
- load i2o_proc module
- login as user and read all entries in /proc/i2o/ directory

My testnode hangs when I try to read any file from /proc/i2o/iop0/030/
directory: I have the shell prompt and even can try to start any new command
which hangs due exec is not works.
Node is pingable, but I cannot login to it nor via ssh neither from local
console. Magic Sysrq keys are works. Kernel space software watchdog module
works OK. But all the new commnds hangs, looks like i2o controller is in coma.

Greg KH wrote:
And I'd classify this a "low" security issue, as you have to be root to
load the i2o_proc module, and I doubt that the distros automatically
load it.

Markus Lidel:
The problem is not really driver related, IIRC. The driver properly send
a I2O command to the controller, but this one couldn't handle it and
instead of just aborting the command "panic's". IIRC it's only Adaptec
related, the Promise controllers doesn't show this behaviour.

Thank you,
 Vasily Averin

Vasily Averin wrote:
> Reading from some i2o related proc files can lead to the i2o controller hang due
> unknown reasons. As a workaround this patch changes the permission of these
> files to root-only accessible.

>
> Signed-off-by: Vasily Averin <vvs@sw.ru>
>
> --- lk2.6/drivers/message/i2o/i2o_proc.c
> +++ lk2.6/drivers/message/i2o/i2o_proc.c
> @@ -1855,17 +1855,17 @@ static i2o_proc_entry i2o_proc_generic_i
> * Device specific entries
> */
> static i2o_proc_entry generic_dev_entries[] = {
> - {"groups", S_IFREG | S_IRUGO, &i2o_seq_fops_groups},
> - {"phys_dev", S_IFREG | S_IRUGO, &i2o_seq_fops_phys_device},
> - {"claimed", S_IFREG | S_IRUGO, &i2o_seq_fops_claimed},
> - {"users", S_IFREG | S_IRUGO, &i2o_seq_fops_users},
> - {"priv_msgs", S_IFREG | S_IRUGO, &i2o_seq_fops_priv_msgs},
> - {"authorized_users", S_IFREG | S_IRUGO, &i2o_seq_fops_authorized_users},
> - {"dev_identity", S_IFREG | S_IRUGO, &i2o_seq_fops_dev_identity},
> - {"ddm_identity", S_IFREG | S_IRUGO, &i2o_seq_fops_ddm_identity},
> - {"user_info", S_IFREG | S_IRUGO, &i2o_seq_fops_uinfo},
> - {"sgl_limits", S_IFREG | S_IRUGO, &i2o_seq_fops_sgl_limits},
> - {"sensors", S_IFREG | S_IRUGO, &i2o_seq_fops_sensors},
> + {"groups", S_IFREG | S_IRUSR, &i2o_seq_fops_groups},
> + {"phys_dev", S_IFREG | S_IRUSR, &i2o_seq_fops_phys_device},
> + {"claimed", S_IFREG | S_IRUSR, &i2o_seq_fops_claimed},
> + {"users", S_IFREG | S_IRUSR, &i2o_seq_fops_users},
> + {"priv_msgs", S_IFREG | S_IRUSR, &i2o_seq_fops_priv_msgs},
> + {"authorized_users", S_IFREG | S_IRUSR, &i2o_seq_fops_authorized_users},
> + {"dev_identity", S_IFREG | S_IRUSR, &i2o_seq_fops_dev_identity},
> + {"ddm_identity", S_IFREG | S_IRUSR, &i2o_seq_fops_ddm_identity},
> + {"user_info", S_IFREG | S_IRUSR, &i2o_seq_fops_uinfo},
> + {"sgl_limits", S_IFREG | S_IRUSR, &i2o_seq_fops_sgl_limits},
> + {"sensors", S_IFREG | S_IRUSR, &i2o_seq_fops_sensors},
> {NULL, 0, NULL}
> };
>

Subject: Re: [patch i2o 1/6] i2o_cfg_passthru cleanup
Posted by Alan Cox on Tue, 15 May 2007 16:42:51 GMT
View Forum Message <> Reply to Message

> - cleanup:
> +cleanup:
> kfree(reply);
> + if (msg)
> +out:
> + i2o_msg_nop(c, msg);

Put the label before the if. Much saner that way

> kfree(reply);
> + if (msg)
> +out:
> + i2o_msg_nop(c, msg);
> return rcode;

Ditto

Subject: Re: [patch i2o 5/6] i2o_proc files permission
Posted by Alan Cox on Tue, 15 May 2007 16:45:33 GMT
View Forum Message <> Reply to Message

On Tue, 15 May 2007 16:47:05 +0400
Vasily Averin <vvs@sw.ru> wrote:

> Reading from some i2o related proc files can lead to the i2o controller hang due
> unknown reasons. As a workaround this patch changes the permission of these
> files to root-only accessible.

I guess you have a crap controller in this case.

This isn't the right fix: Detect your specific buggy control and do not
register the problem /proc nodes for it at all.

Alan

Subject: Re: [patch i2o 6/6] i2o debug output cleanup
Posted by Alan Cox on Tue, 15 May 2007 16:46:25 GMT
View Forum Message <> Reply to Message

On Tue, 15 May 2007 16:48:16 +0400
Vasily Averin <vvs@sw.ru> wrote:

> fixed output of i2o debug messages, extra KERN_ are removed
>

Otherwise looks good. ACK apart from the interesting goto targets and
the /proc handling.

Subject: Re: [patch i2o 5/6] i2o_proc files permission

Posted by vaverin on Wed, 16 May 2007 04:58:37 GMT
View Forum Message <> Reply to Message

Alan Cox wrote:
> On Tue, 15 May 2007 16:47:05 +0400
> Vasily Averin <vvs@sw.ru> wrote:
>
>> Reading from some i2o related proc files can lead to the i2o controller hang due
>> unknown reasons. As a workaround this patch changes the permission of these
>> files to root-only accessible.
>
> I guess you have a crap controller in this case.
>
> This isn't the right fix: Detect your specific buggy control and do not
> register the problem /proc nodes for it at all.

Alan,

We have i2o on one of our testnodes: Adaptec Zero channel RAID 2010S integrated
on MSI MS-9136 motherboard and I don't have access to another i2o hardware.
Also I don't have any ideas how to detect buggy i2o node. As far as I understand
Markus is right, and i2o hardware couldn't handle received message that on the
first glance looks correctly.
>From my POV it is bug in firmware and only vendor is able to fix it correctly.
My patch is not a fix but workaround only -- it just do not allow to crash the
node by any user in case when node admin due some reasons has loaded i2o_proc
module.

Thank you,
Vasily Averin

Subject: Re: [patch i2o 5/6] i2o_proc files permission
Posted by Greg KH on Wed, 16 May 2007 09:27:01 GMT
View Forum Message <> Reply to Message

On Tue, May 15, 2007 at 04:59:49PM +0400, Vasily Averin wrote:
> I would add:
> I've reported about this issue some time ago to security@kernel.org
> How this lockup can be reproduced:
> - boot the kernel,
> - load i2o_proc module
> - login as user and read all entries in /proc/i2o/ directory
>
> My testnode hangs when I try to read any file from /proc/i2o/iop0/030/
> directory: I have the shell prompt and even can try to start any new command
> which hangs due exec is not works.
> Node is pingable, but I cannot login to it nor via ssh neither from local

> console. Magic Sysrq keys are works. Kernel space software watchdog module
> works OK. But all the new commnds hangs, looks like i2o controller is in coma.
>
> Greg KH wrote:
> And I'd classify this a "low" security issue, as you have to be root to
> load the i2o_proc module, and I doubt that the distros automatically
> load it.

Yeah, I said it as I didn't see a "simple" way to fix it at the time.
If you have solved this now with this patch, I have no objection to it.

thanks,

greg k-h

Subject: Re: [patch i2o 5/6] i2o_proc files permission
Posted by Alan Cox on Wed, 16 May 2007 12:49:03 GMT
View Forum Message <> Reply to Message

> We have i2o on one of our testnodes: Adaptec Zero channel RAID 2010S integrated
> on MSI MS-9136 motherboard and I don't have access to another i2o hardware.
> Also I don't have any ideas how to detect buggy i2o node. As far as I understand
> Markus is right, and i2o hardware couldn't handle received message that on the
> first glance looks correctly.
> >From my POV it is bug in firmware and only vendor is able to fix it correctly.
> My patch is not a fix but workaround only -- it just do not allow to crash the
> node by any user in case when node admin due some reasons has loaded i2o_proc
> module.

It merely allows it to occur by accident if root does something like find
in the wrong place and it also breaks the expected behaviour for existing
systems.

There are two ways you can identify a specific controller if the vendor
didn't totally screw up

The first is the PCI idents or subidents, the second is to look at the
tables returned when the controller is being set up. In fact the
Adaptecrap controllers (ex DPT) are already detected and we set
c->adaptec in pci.c so we can stop them exploding for other reasons and
to use some extensions they have.

Thus I think the right solution is to skip registering those proc files
(and unregistering them on unload) if c->adaptec is set.
