```
From: imv4bel@gmail.com
To: mchehab@kernel.org
Cc: Hyunwoo Kim <imv4bel@gmail.com>,
        kernel@tuxforce.de, linux-media@vger.kernel.org,
        linux-usb@vger.kernel.org, cai.huoqing@linux.dev, tiwai@suse.de
Subject: [PATCH 3/4] media: dvb-core: Fix use-after-free due to race condition occurring in dvb_register_device()
Date: Tue, 15 Nov 2022 05:18:21 -0800   [thread overview]
Message-ID: <20221115131822.6640-4-imv4bel@gmail.com> (raw)
In-Reply-To: <20221115131822.6640-1-imv4bel@gmail.com>

From: Hyunwoo Kim <imv4bel@gmail.com>

dvb_register_device() dynamically allocates fops with kmemdup()
to set the fops->owner.
And these fops are registered in 'file->f_ops' using replace_fops()
in the dvb_device_open() process, and kfree()d in dvb_free_device().

However, it is not common to use dynamically allocated fops instead
of 'static const' fops as an argument of replace_fops(),
and UAF may occur.
These UAFs can occur on any dvb type using dvb_register_device(),
such as dvb_dvr, dvb_demux, dvb_frontend, dvb_net, etc.

So, instead of kfree() the fops dynamically allocated in
dvb_register_device() in dvb_free_device() called during the
.disconnect() process, kfree() it collectively in exit_dvbdev()
called when the dvbdev.c module is removed.

Signed-off-by: Hyunwoo Kim <imv4bel@gmail.com>
---
 drivers/media/dvb-core/dvbdev.c | 83 +++++++++++++++++++++++++---------
 include/media/dvbdev.h          | 15 ++++++
 2 files changed, 77 insertions(+), 21 deletions(-)

diff --git a/drivers/media/dvb-core/dvbdev.c b/drivers/media/dvb-core/dvbdev.c
index 675d877a67b2..424cf92c068e 100644
--- a/drivers/media/dvb-core/dvbdev.c
+++ b/drivers/media/dvb-core/dvbdev.c
@@ -27,6 +27,7 @@
 #include <media/tuner.h>

 static DEFINE_MUTEX(dvbdev_mutex);
+static LIST_HEAD(dvbdevfops_list);
 static int dvbdev_debug;

 module_param(dvbdev_debug, int, 0644);
@@ -448,14 +449,15 @@ int dvb_register_device(struct dvb_adapter *adap, struct dvb_device **pdvbdev,
                        enum dvb_device_type type, int demux_sink_pads)
 {
        struct dvb_device *dvbdev;
-       struct file_operations *dvbdevfops;
+       struct file_operations *dvbdevfops = NULL;
+       struct dvbdevfops_node *node, *new_node;
        struct device *clsdev;
        int minor;
        int id, ret;

        mutex_lock(&dvbdev_register_lock);

-       if ((id = dvbdev_get_free_id (adap, type)) < 0){
+       if ((id = dvbdev_get_free_id (adap, type)) < 0) {
                mutex_unlock(&dvbdev_register_lock);
                *pdvbdev = NULL;
                pr_err("%s: couldn't find free device id\n", __func__);
@@ -463,18 +465,45 @@ int dvb_register_device(struct dvb_adapter *adap, struct dvb_device **pdvbdev,
        }

        *pdvbdev = dvbdev = kzalloc(sizeof(*dvbdev), GFP_KERNEL);
-
        if (!dvbdev){
                mutex_unlock(&dvbdev_register_lock);
                return -ENOMEM;
        }

-       dvbdevfops = kmemdup(template->fops, sizeof(*dvbdevfops), GFP_KERNEL);
+       /*
+        * When a device of the same type is probe()d more than once,
+        * the first allocated fops are used. This prevents memory leaks
+        * that can occur when the same device is probe()d repeatedly.
+        */
+       list_for_each_entry(node, &dvbdevfops_list, list_head) {
+               if (node->fops->owner == adap->module &&
+                               node->type == type &&
+                               node->template == template) {
+                       dvbdevfops = node->fops;
+                       break;
+               }
+       }

-       if (!dvbdevfops){
-               kfree (dvbdev);
-               mutex_unlock(&dvbdev_register_lock);
-               return -ENOMEM;
+       if (dvbdevfops == NULL) {
+               dvbdevfops = kmemdup(template->fops, sizeof(*dvbdevfops), GFP_KERNEL);
+               if (!dvbdevfops) {
+                       kfree(dvbdev);
+                       mutex_unlock(&dvbdev_register_lock);
+                       return -ENOMEM;
+               }
+
+               new_node = kzalloc(sizeof(struct dvbdevfops_node), GFP_KERNEL);
+               if (!new_node) {
+                       kfree(dvbdevfops);
+                       kfree(dvbdev);
+                       mutex_unlock(&dvbdev_register_lock);
+                       return -ENOMEM;
+               }
+
+               new_node->fops = dvbdevfops;
+               new_node->type = type;
+               new_node->template = template;
+               list_add_tail (&new_node->list_head, &dvbdevfops_list);
        }

        memcpy(dvbdev, template, sizeof(struct dvb_device));
@@ -484,20 +513,20 @@ int dvb_register_device(struct dvb_adapter *adap, struct dvb_device **pdvbdev,
        dvbdev->priv = priv;
        dvbdev->fops = dvbdevfops;
        init_waitqueue_head (&dvbdev->wait_queue);
-
        dvbdevfops->owner = adap->module;
-
        list_add_tail (&dvbdev->list_head, &adap->device_list);
-
        down_write(&minor_rwsem);
 #ifdef CONFIG_DVB_DYNAMIC_MINORS
        for (minor = 0; minor < MAX_DVB_MINORS; minor++)
                if (dvb_minors[minor] == NULL)
                        break;
-
        if (minor == MAX_DVB_MINORS) {
+               if (new_node) {
+                       list_del (&new_node->list_head);
+                       kfree(dvbdevfops);
+                       kfree(new_node);
+               }
                list_del (&dvbdev->list_head);
-               kfree(dvbdevfops);
                kfree(dvbdev);
                up_write(&minor_rwsem);
                mutex_unlock(&dvbdev_register_lock);
@@ -506,41 +535,46 @@ int dvb_register_device(struct dvb_adapter *adap, struct dvb_device **pdvbdev,
 #else
        minor = nums2minor(adap->num, type, id);
 #endif
-
        dvbdev->minor = minor;
        dvb_minors[minor] = dvbdev;
        up_write(&minor_rwsem);
-
        ret = dvb_register_media_device(dvbdev, type, minor, demux_sink_pads);
        if (ret) {
                pr_err("%s: dvb_register_media_device failed to create the mediagraph\n",
                      __func__);
-
+               if (new_node) {
+                       list_del (&new_node->list_head);
+                       kfree(dvbdevfops);
+                       kfree(new_node);
+               }
                dvb_media_device_free(dvbdev);
                list_del (&dvbdev->list_head);
-               kfree(dvbdevfops);
                kfree(dvbdev);
                mutex_unlock(&dvbdev_register_lock);
                return ret;
        }

-       mutex_unlock(&dvbdev_register_lock);
-
        clsdev = device_create(dvb_class, adap->device,
                               MKDEV(DVB_MAJOR, minor),
                               dvbdev, "dvb%d.%s%d", adap->num, dnames[type], id);
        if (IS_ERR(clsdev)) {
                pr_err("%s: failed to create device dvb%d.%s%d (%ld)\n",
                      __func__, adap->num, dnames[type], id, PTR_ERR(clsdev));
+               if (new_node) {
+                       list_del (&new_node->list_head);
+                       kfree(dvbdevfops);
+                       kfree(new_node);
+               }
                dvb_media_device_free(dvbdev);
                list_del (&dvbdev->list_head);
-               kfree(dvbdevfops);
                kfree(dvbdev);
                return PTR_ERR(clsdev);
        }
+
        dprintk("DVB: register adapter%d/%s%d @ minor: %i (0x%02x)\n",
                adap->num, dnames[type], id, minor, minor);

+       mutex_unlock(&dvbdev_register_lock);
        return 0;
 }
 EXPORT_SYMBOL(dvb_register_device);
@@ -569,7 +603,6 @@ void dvb_free_device(struct dvb_device *dvbdev)
        if (!dvbdev)
                return;

-       kfree (dvbdev->fops);
        kfree (dvbdev);
 }
 EXPORT_SYMBOL(dvb_free_device);
@@ -1061,9 +1094,17 @@ static int __init init_dvbdev(void)

 static void __exit exit_dvbdev(void)
 {
+       struct dvbdevfops_node *node, *next;
+
        class_destroy(dvb_class);
        cdev_del(&dvb_device_cdev);
        unregister_chrdev_region(MKDEV(DVB_MAJOR, 0), MAX_DVB_MINORS);
+
+       list_for_each_entry_safe(node, next, &dvbdevfops_list, list_head) {
+               list_del (&node->list_head);
+               kfree(node->fops);
+               kfree(node);
+       }
 }

 subsys_initcall(init_dvbdev);
diff --git a/include/media/dvbdev.h b/include/media/dvbdev.h
index 2f6b0861322a..1e5413303705 100644
--- a/include/media/dvbdev.h
+++ b/include/media/dvbdev.h
@@ -187,6 +187,21 @@ struct dvb_device {
        void *priv;
 };

+/**
+ * struct dvbdevfops_node - fops nodes registered in dvbdevfops_list
+ *
+ * @fops:              Dynamically allocated fops for ->owner registration
+ * @type:              type of dvb_device
+ * @template:          dvb_device used for registration
+ * @list_head:         list_head for dvbdevfops_list
+ */
+struct dvbdevfops_node {
+       struct file_operations *fops;
+       enum dvb_device_type type;
+       const struct dvb_device *template;
+       struct list_head list_head;
+};
+
 /**
  * dvb_register_adapter - Registers a new DVB adapter
  *
--
2.25.1
```

Thread overview: 6+ messages / expand[flat|nested] mbox.gz Atom feed top

```
2022-11-15 13:18 [PATCH 0/4] Fix multiple race condition vulnerabilities in dvb-core and device driver imv4bel
2022-11-15 13:18 ` [PATCH 1/4] media: dvb-core: Fix use-after-free due to race condition occurring in dvb_frontend imv4bel
2022-11-15 13:18 ` [PATCH 2/4] media: dvb-core: Fix use-after-free due to race condition occurring in dvb_net imv4bel
2022-11-15 13:18 ` imv4bel [this message]
2022-11-17  4:16   ` [PATCH 3/4] media: dvb-core: Fix use-after-free due to race condition occurring in dvb_register_device() Dan Carpenter
2022-11-15 13:18 ` [PATCH 4/4] media: ttusb-dec: Fix memory leak in ttusb_dec_exit_dvb() imv4bel
```

find likely ancestor, descendant, or conflicting patches for this message:

dfblob:675d877a67b dfblob:2f6b0861322 dfblob:424cf92c068
dfblob:1e541330370

search (help)

Reply instructions:

You may reply publicly to this message via plain-text email
using any one of the following methods: