TP : CAESAR CIPHERING DEVICE

Goal : This TP aims to build and virtualize a caesar ciphering device used to encrypt or decrypt the data that is passed to it.

This work should be done using a linux operating system. you need to install QEMU for we are going to use it to virtualize our device. think about installing all the necessary tools for kernel development. here is a list of the thinks you need to download for this work :

- packages for kernel development (if you have not done it yet):
  run in this order (for Ubuntu systems) :
  ```
  $  sudo apt-get build-dep linux linux-image-$(uname -r)
  $ sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm ninja-build libglib2.0-dev libpixman-1-dev qemu-kvm
  ```
- Guest kernel download :
  https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.4.214.tar.xz
- QEMU download and build instructions (version 7.1.0 in this case) :
  https://www.qemu.org/download/. This git contains an initramfs where to boot QEMU from if needed. use it with the -initrd attribute.

PRELIMINARY TASKS

We need to configure the kernel that we are going to use for the virtual machine. Since it takes a while to compile the first time, we will launch the compilation first of all and we will come back later for the next steps.

- type the following commands to compile it
  ```
  $ cp -v /boot/config-$(uname -r) .config
  $ yes "" | make oldconfig
  $ make vmlinux
  ```
  # this command builds the vmlinux file that we are going to use; use the -j option for multicore compilation *
- using the provided disk image in the disk folder and the vmlinux file to run the virtual machine using the file named launch-vm.sh script.

* : If you run into the error containing : No rule to make target 'debian/canonical-certs.pem', in your .config file, change put the empty value (” ”) to the line CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem".

Once you are able to launch a VM; all that remains to do after each modification of the guest kernel is to compile it and reboot the virtual machine.

1. CAESAR CIPHERING DEVICE DRIVER

This section aims to build a device driver that performs the caesar ciphering. here is the caesar ciphering principle:

“*To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by a given integer key k. Each letter is replaced by the letter k letters ahead in the alphabet. if k is three, A becomes D, B becomes E, and so on. For the last letters, we can think of the alphabet as a circle and "wrap around*".

We should be able to :

- encrypt a text using a key
- decrypt a text using a key
- change the key’s value

This device has two attributes which are :

- key : which is the value by which we shift the letters.
- size : that is the limit of the text that can be encrypted.

1.1. DEVICE INITIALIZATION

The file caesar-dev.c is the skeleton where you will add your code.

1) Initialize the character device, by following the different steps seen during the course :
   - memory region allocation
   - major number initialization
   - sysfs registration
   - character device initialization
   - device node création

   You need to fill the function caesardev_init for this task.
2) The sysfs class contains one attribute dev_uevent that allows us to change the permissions of the function. In the context of this work we want every user to be able to read or write. Thus pass the function called *caesardev_uevent* to that attribute.

3) Complete the function caesardev_destroy to destroy the device when it is no longer needed. do not forget to unregister the sysfs class.

We can now compile the device driver and test it. use the following command to make and install the driver.

```
$ make && sudo insmod caesar-dev.ko
```

Warning : If your architecture is not x86, feel free to modify the makefile

At this step, we have finished with the init and destroy functions we are now going to build the interface with the userspace.

You might see the sysfs class which has been created for caesardev by typing :

```
$ tree /sys/devices/virtual/caesardev/
```

About the permissions on the device, you might use the command :

```
$ ls -l /dev/caesardev to see them
```

Finally you can read, write data to that file and use dmesg to see the results.

1.2. USERSPACE INTERFACE

This section will help us to build a module that allows us to communicate properly with the userspace. As there will be some data exchange. Consider using the function :

unsigned long copy_to_user(void __user *to, const void *from, unsigned long n);

and

unsigned long copy_from_user(void *to, const void __user *from, unsigned long n);

when it comes to transfer some text data.

We are going to set the device attributes key and size for the userspace to be able to read or set them.

1) use the macro **DEVICE_ATTR_RW(name)** to define the two attributes. where name is the attribute name, the mode is set to read and write with the value S_IWUSR | S_IRUGO. This macro requires the definition of two functions named **name_show** and **name_store** functions which are called when a get or set respectively will be done from the userspace.
2) write the show and store functions for the two parameters. do not forget to add the parameters into the device data struct. In order to convert a string into a uint32_t, you can use the function **kstrtou32(const char *s, unsigned int base, u32 *res)** (use base = 0).

   For the store functions, save the data to the device data struct.

3) Once the attributes have been declared with their callbacks, we should create the attribute group to register them all at once. This step has been done for you. What you need to do now, is to register the attribute group in the **caesardev_class->dev_groups** lvalue. This will tell sysfs that these attributes should be registered in the system. you can use the command tree as earlier to ensure that your attribute files have been created correctly.
4) You can now read these files using the **cat** command in the shell or set the attributes value by using the `$ echo <value> | sudo tee <path>.`

The sysfs interface is thus useful to set the device parameters. we will now write the callbacks for the file in the /dev folder for us to be able to send a text to be encrypted. There are two useful callbacks which are **read** and **write**.

5) Write the write callback. There are two possible cases :
   a) We write the direction of the operations (see the comments on the device data struct for more info). This direction is used for all operations until a new direction is set
   b) We write the text to be processed and perform the operations according to the direction. Make sure that it doesn't exceed the limit. in that case, return an error.
6) Write the read callback. This callback is used to get the results of the operation performed in the last write.
   to test the read function using the valid number of bytes, use the command : `$ head -c<value> <path to file>.` Where value is the number of bytes to be read.
7) you can now compile and test your driver.

Note : The ciphering operation is done in the function :
ssize_t caesar_cipher(char * message, char *output). It takes as input the message that needs to be processed according to the direction, the output array where the result will be written to, and outputs the size of the processed text or -1 if there was an error.

2. VIRTIO DEVICE DRIVER : VIRTIO_CAESAR_DEV

Now that we have a working driver, the goal is to be able to use it from inside a virtual machine directly without emulating it. To do that, we will write a virtio driver associated with that driver. In the preliminary work, you have set up a virtual machine using QEMU and a guest kernel. the virtio device driver will be inserted in the guest kernel, and the virtio driver will be written inside QEMU.

2.1. VIRTIO DEVICE SPECIFICATIONS

The first step in writing a virtio driver is to define the virtio device specifications. They drive the way the driver and the device will be built. According to the real caesar device from the first part, define the following virtio specifications in a text file in the tp-git folder:

1. Device usage : define the usage of the virtio device
2. Device ID : define the virtio ID of the device. be careful not to use an already taken virtio ID
3. Virtqueues : define the number of virtqueues you need for this driver and a description of each one.
4. Feature bits : Define the features list if required

You might also define the other specifications if relevant for the current driver.

2.2. VIRTIO DRIVER

This part consists of writing the virtio device driver. The driver has some parts wħich are described as follows :

- The probe : THe probe is the function that initializes the driver and the driver's information according to the device it is linked to.
- The irq handlers definition : that handles the interruptions done by qemu,
- The sysfs interface : if required
- The other functions.

The task in this section is to write a virtio driver that :

- exposes the exact same interface to the userspace than the caesar device does.
- Is able to modify the underlying parameters of the real caesar device from the virtual machine
- Perform the encryption / Decryption with the same constraints
- That follows the device specifications that have been described earlier.

2.3. VIRTIO DEVICE

This part contains 2 elements :
- The caesar device emulated by qemu that receives the requests from the guest OS and performs them.
- The bus wrapper for the caesar device. You must work with the PCI wrapper.

The work here consists of writing the two parts of the virtio device. You can inspire yourself from the virtio_example that we have seen during the course. You can inspire yourself from the virtio example that we have seen during the course.
NB : If you choose to use the version of QEMU that is here : https://github.com/ybettan/qemu.git, be careful, the twø are not compiled the same way.

2.4. ASSEMBLE IT

As the device and its driver are ready at this step, you can run the virtual machine and add the parameter -device <name of your PCI device> to your qemu command. That attribute is here to tell QEMU that you want to run your virtual machine, using that device.