LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

External / Untrusted Device Management

- Rajat Jain
  Google

Google

Trust v/s Location - Current Situation

- Firmware marks certain PCIe rootports as “External-facing”.
- PCI subsystems marks any devices downstream those as “Untrusted”

pci_dev->untrusted is used:

- By PCI to disallow ATS (and hopefully block translated addresses).
- By Intel IOMMU driver to setup bounce buffers + enforce “strictness” of IOMMU

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Separating Trust from Location

- Uncovered use cases today
  - Internal untrusted devices
  - Only a subset of devices in a hierarchy untrusted
- Need a way to mark devices as untrusted, independent of location.
  - Info could come from user (Cmdline / sysfs)
  - Or platform (firmware / acpi / devicetree - will need a new property)
- Should have 2 separate fields => location and untrusted.
  - kernel components can use these as needed
- At what layer is it helpful to have these? (device core v/s PCI)

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

RFC #1: Expose “Location” to userspace

LOCATION

- Semantically an immutable property of the device.
  - Primary current interest in current context is “internal vs external”.
  - dev->location filled by bus when registering a device.
- Initialization options:
  - From platform ACPI/DT info (which we can only hope is correct).
  - Override using cmdline
  - Override via sysfs (discussed in RFC #3) - has challenges.
- Userspace use cases (+ may be kernel use cases we don’t know yet?)
  - statistics for external devices
  - show something in UI when an external device connects etc.
  - Identifying location seems like a useful piece of info.

RFC #2: Expose “Untrusted” to userspace

- Semantically represents a knob that defines trustworthiness of a device.
- Initialization options:
    - From platform ACPI/DT info
        - May be use location, just like we do today (to not cause regressions)?
        - Introduce a property for firmware to mark devices as trusted/untrusted?
    - Override using cmdline
    - Override via sysfs (discussed in RFC #3) - has challenges.
- Expose it to userspace to allow the userspace to implement userspace policies it may want for untrusted devices:
    - Warn about an untrusted device plugged in.
    - Log device VID/DID/serial number etc for the admin.
    - Control whether to attach a driver.
- Struct device->untrusted v/s struct pci_device->untrusted ?

RFC #3: Make “Location” and “Untrusted” writable in sysfs

- No good solution for devices in boot path
  - No userspace to tell if a device is trusted or not.
  - Need to rely on info provided by firmware (or command line).

- “Untrusted” need to be consumed by early PCI / IOMMU drivers, atleast before its dri begins to attach.

- Solution would entail to allow a mechanism that would:
  - Enumerate devices, but stop before attaching a driver to the device.
  - Let userspace change “Location” & “untrusted” as and if needed.
  - A way for userspace to say go ahead and attach the drivers
    - (just before which, PCI/IOMMU need to now consume the fields).

- This problem may be more important to solve for “untrusted” attribute than the “location” (only use case: buggy firmware).

cont...

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

… RFC #3 (Options Considered)

Option 1: USB's "authorized / default_authorized " UAPI fits the bill to a large degree semantically, and moving it to driver core sounds good, but there are some challenges:

- USB "authorized" UAPI conflicts with thunderbolt's "authorized".
- If a device is deauthorized, need to also deauthorize all child devices? Authorization for a new hierarchy may be tricky / racy for userspace.
- Current USB UAPI is widely used, and I'm worried that we might need some changes (to adapt to new needs). (Bonus slides if needed)
- May need a much wider discussion and buy in (Thunderbolt? User space tools?) (and effort)? Need suggestions and help to finalize the UAPI.

Option 2: Use "drivers_autoprobe" to disable autoprobe of devices (for <some devices=?>), and enable later, via "drivers_probe".

- "drivers_autoprobe" doesn't work for PCI because it decided to attach the drivers explicitly via device_attach() (which does not honour drivers_autoprobe).

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Requests

- Value in starting with something (RFC #1 and RFC #2) even though solution to RFC #3 is not very clear?
  - They don't have a functional change, but provide info to user who today has no way to know the kernel's knowledge about which devices it is choosing to trust.

- Looking for support / help
  - Volunteers to own pieces!
  - Alternate Big Picture proposals

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Thank You !

Open Discussion / Questions

Google

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

… RFC #3 (Option1: USB “authorized” UAPI)

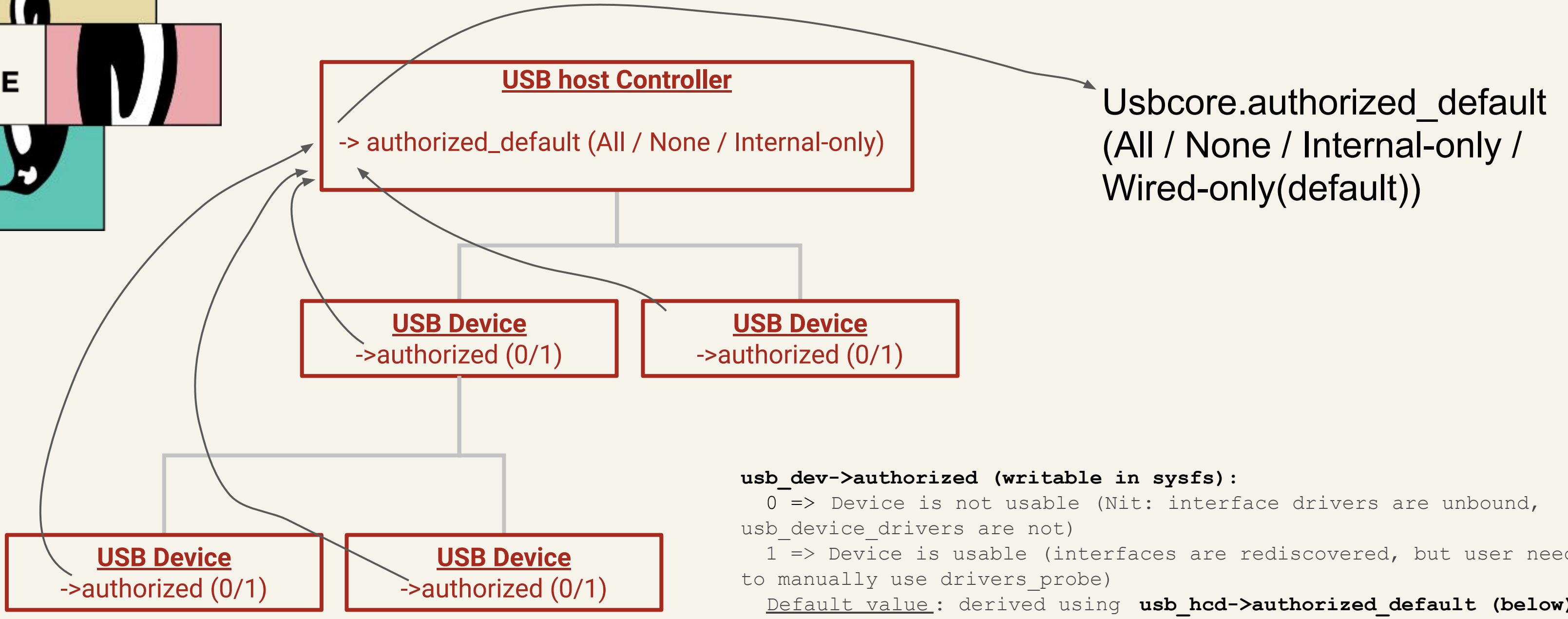
Current Semantics:

usb_dev->authorized (writable in sysfs):
  0 => Device is not usable (Nit: interface drivers are unbound, usb_device_drivers are not)
  1 => Device is usable (interfaces are rediscovered, but user needs to manually use drivers_probe)
  Default value : derived using **usb_hcd->authorized_default (below)** and “location determined in bus specific way”

usb_hcd->authorized_default (writable in sysfs):
  USB_DEVICE_AUTHORIZE_NONE => All FUTURE devices below this hcd will be deauthorized by default
  USB_DEVICE_AUTHORIZE_ALL => All FUTURE devices below this hcd will be authorized by default
  USB_DEVICE_AUTHORIZE_INTERNAL => Only internal FUTURE devices below this will be authorized by default
  Default value : derived using **usbcore’s authorized_default** parameter and “hcd specific info (wired/wireless)”

usbcore’s authorized_default parameter (writable in sysfs):
  USB_AUTHORIZED_NONE => Set any new Future usb_hcd’s authorized_default to USB_DEVICE_AUTHORIZE_NONE
  USB_AUTHORIZED_ALL => Set any new Future usb_hcd’s authorized_default to USB_DEVICE_AUTHORIZE_ALL
  USB_AUTHORIZED_INTERNAL => Set any new Future usb_hcd’s authorized_default to USB_DEVICE_AUTHORIZE_INTERNAL
  USB_AUTHORIZED_WIRED => Set any new Future usb_hcd’s authorized_default to either NONE / All depending on wired/wireless
  Default value : USB_AUTHORIZED_WIRED

Current USB "authorized" UAPI



```
usb_dev->authorized (writable in sysfs):
  0 => Device is not usable (Nit: interface drivers are unbound,
usb_device_drivers are not)
  1 => Device is usable (interfaces are rediscovered, but user needs
to manually use drivers_probe)
  Default value : derived using  usb_hcd->authorized_default (below)
and "location determined in bus specific way"
```

```
usb_hcd->authorized_default (writable in sysfs):
  USB_DEVICE_AUTHORIZE_NONE =>  All FUTURE devices below this hcd will be deauthorized by default
  USB_DEVICE_AUTHORIZE_ALL =>  All FUTURE devices below this hcd will be authorized by default
  USB_DEVICE_AUTHORIZE_INTERNAL =>  Only internal FUTURE devices below this will be authorized by default
  Default value : derived using  usbcore's authorized_default parameter and "hcd specific info (wired/wireless)"
```

“Authorized” UAPI Challenges for our use case

- Conflicts with Thunderbolt.
- Is “Location based” (USB_DEVICE_AUTHORIZE_INTERNAL)

- Where do we hold “authorized_default” in device core? (USB has it at usb_hcd level).
  - May be have it at each device level that controls its children?

- Nit: usb_dev->authorized=0 only unbinds interface drivers (usb_driver), but not usb_device_driver, but I think it may be OK to unbind them also with this UAPI?

- USB also has interface->default, usb_hcd->interface_authorized_default. That will stay in USB. OK?

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Device Core “authorized” UAPI

- Bus drivers to populate up dev->untrusted and dev->location before device_add().
- Bus drivers to provide hooks to device core:
  - authorize_device()
  - deauthorize_device()
  - authorized_default()

device->authorized (writable in sysfs):

0 => Device is not usable (Device core unbinds the drivers, calls bus_type specific deauthorize_device() hook)

1 => Device is usable (Device core calls bus_type specific authorize_device() hook, (and then possibly autobinds the drivers)?)

Default value: derived by device core using **dev->parent->authorized_default (below)** and **dev->location OR dev->untrusted**

device->authorized_default (writable in sysfs):

DEVICE_AUTHORIZE_NONE => All FUTURE devices below this dev will be deauthorized by default
DEVICE_AUTHORIZE_ALL => All FUTURE devices below this dev will be authorized by default
DEVICE_AUTHORIZE_INTERNAL => Only internal FUTURE devices below this dev will be authorized by default
DEVICE_AUTHORIZE_TRUSTED => Only trusted FUTURE devices below this dev will be authorized by default
Default value: derived using bus_type’s authorized_default(dev) hook ”

Bus can have its have its own policy to implement **authorized_default()** policy (writable in sysfs). Eg USB can continue to have a parameter with values:

USB_AUTHORIZED_NONE => authorized_default() returns DEVICE_AUTHORIZE_NONE
USB_AUTHORIZED_ALL => authorized_default() returns DEVICE_AUTHORIZE_ALL
USB_AUTHORIZED_INTERNAL => authorized_default() returns DEVICE_AUTHORIZE_INTERNAL
USB_AUTHORIZED_WIRED => authorized_default() returns NONE / All depending on wired/wireless
Default value: USB_AUTHORIZED_WIRED