DRI-next/DRM2: A walkthrough the Linux Graphics stack and its security

Overview, vulnerabilities, attacks and discussions around the graphic stack's security

Martin Peres & Timothée Ravier

Ph.D. student at LaBRI, System security engineer

Brussels, 2 & 3 February 2013, FOSDEM

Summary

Introduction : Important properties

Wanted properties

- Enable users to interact with applications (Input management);
- Enable applications to display content on one or several screens.

Applications should not be able to (wrt other applications)

- eavesdrop : Confidentiality;
- tamper : Integrity;
- deny service: Availability.

Security Use Case : Confidentiality

Use cases 0 & 1

- The user is shopping online ;
- He/she keys in the credit card number;
- A keylogger was installed on the computer
  or
  A program takes periodical screenshots;
- His/her credit card number got stolen!

Security Use Case : Integrity

Use case

- The user is visiting his bank's website;
- He/she checks the website address (https + right domain);
- He/she is unaware that he/she is visiting a fake website and that Firefox's address bar has been redrawn by a malware;
- His/her bank information got stolen!

Security Use Case : Availability

User's expectation towards availability

- Users think their computers do multitasking;
- Thus, one app shouldn't be able to bring the system down;
- $\Rightarrow$ Applications should never be able to deny access to other applications.

Example

- Screen locks

Summary

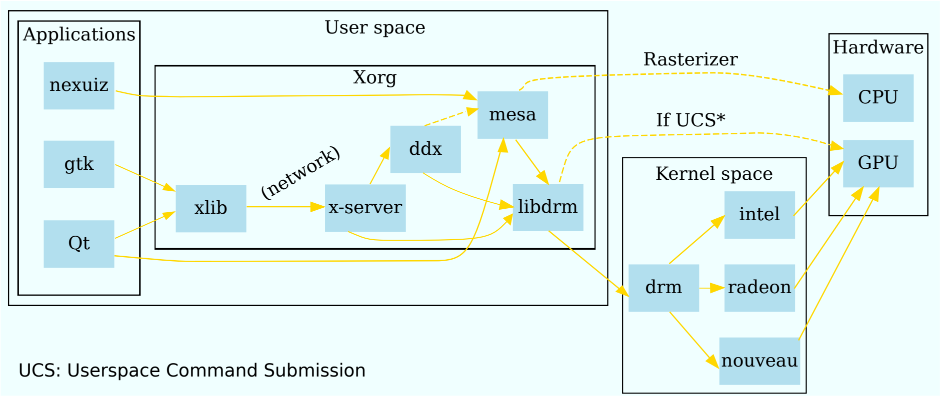
Overview of the Linux graphics stack

Related hardware

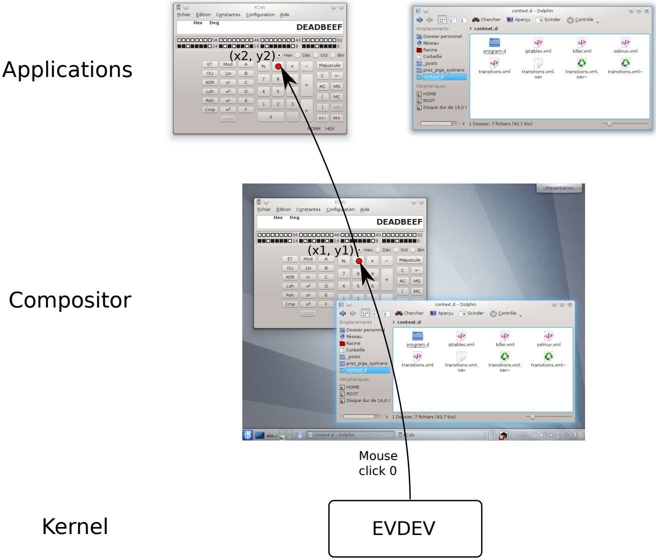
- Desktops and Laptops;
- Smartphones and embedded boards.

Related software on Linux

- Linux kernel / DRM (Direct Rendering Manager);
- X-Server, Wayland/Weston;
- Mesa, cairo...;
- Toolkits: Qt, GTK+, EFL...


User space
Applications
nexuiz
gtk
Qt
Xorg
xlib
(network)
x-server
ddx
mesa
libdrm
Rasterizer
If UCS*
Hardware
CPU
GPU
Kernel space
intel
drm
radeon
nouveau
UCS: Userspace Command Submission

Input Management as it should be (Wayland)



Input Management - Security flaws

X11

- Any client can be a virtual keyboard : input injection;
- Any client can become a keylogger;
- Many apps rely on these properties $->$ unfixable.

Wayland

- Same as X11 but fixable because there are no apps yet.

DRM : Direct Rendering Manager

DRM : Direct Rendering Manager

- Inits and configures the GPU;
- Performs Kernel Mode Setting (KMS);
- Exports privileged GPU primitives:
  - Create context + VM allocation;
  - Command submission;
  - VRAM memory management: GEM & TTM;
  - Buffer-sharing: GEM & DMA-Buf;
- Implementation is driver-dependent.

libDRM

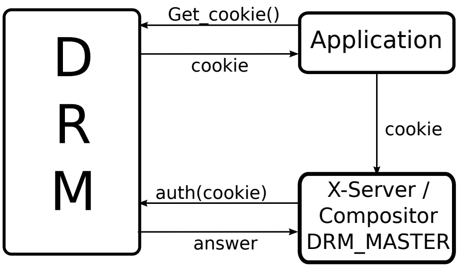
- Wraps the DRM interface into a usable API;
- Factors-out some code;
- Is meant to be only used by Mesa & the DDX;

DRM : Authentication

DRM : Authentication

- The DRM master has all the rights (modeset + GPU usage);
- It can also authenticate clients;
- Only authenticated clients can use the GPU.
- → No non-privileged apps (even for GPGPU).

DRM Auth



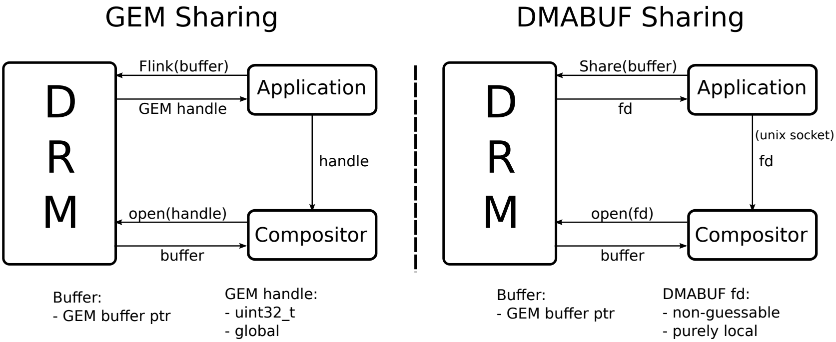
Who can be the DRM master?

- Needs to have the CAP_SYS_ADMIN (root);
- One master at a time;
- The first one to request it gets it;
- The DRM master rights can be released.

DRM2 / Render nodes : Spliting the DRM rights

- Need for head-less apps (video en/decoding, GPGPU);
- Drivers should be able to expose a safe subset of operations:
  - Request contexts;
  - Buffer allocation;
  - Command submission;
  - Buffer sharing*.
- Keep modesetting for DRM_MASTERs.

Buffer-Sharing: GEM vs DMABUF



Buffer-Sharing: GEM vs DMABUF

Graphics Execution Manager (GEM) buffer sharing

- The standard interface for creating, sharing, mapping and modifying buffers;
- Allow buffer sharing: flink() to share (returns a guessable ID), open() to open a shared buf;
- Deprecated by DMA-Buf.

DMA-Buf

- GEM flink is unsecure (poor access control once flinked);
- GEM flink doesn't work across drivers (no optimus support!);
- DMA-Buf solves the latter and uses file descriptors to identify shared buffers;
- This fd can then be passed on to a process using unix sockets;
- The requesting process is responsible for sending the buffer.

DRM2 / Render Nodes

DRM2 / Render Nodes

- First patchset by David Airlie, second by Kristian Høgsberg;
- I then proposed a patchset for the userspace (Nouveau-only):
  - libdrm: Associate the “normal” and the “render” node;
  - dri2proto: Add a “requiresAuth” parameter;
  - XServer: Expose the requireAuth parameter to the ddx;
  - Nouveau DDX: Use the requireAuth parameter;
  - Mesa: Open the right device + authenticate if needed.

The TODO list

- Fix DRM’s BO mmaping address from per-device to per-fd;
- Rework the patches and include more drivers;
- Test every combination to check no regression can happen;
- Port to Wayland/Weston.

Application $< -- >$ XServer communication

Application $< -- >$ XServer communication

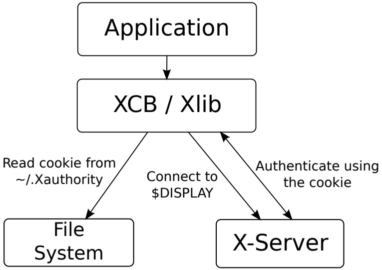
- XLib: Traditionnal X communication medium;
- DRI2: Communication between direct GL apps and the XServer, buffer sharing using GEM;
- XSHM: Use SHM instead of copying data in the XLib's socket.

Buffer management

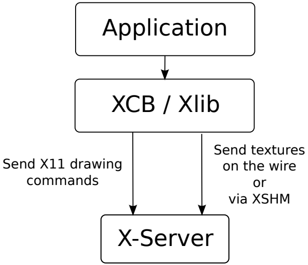
- A client requests a XDrawable allocation via Xlib/XCB/DRI2;
- The X-server allocates the buffer an ID;
- A client can start rendering inside the buffer.

Rendering 2D applications

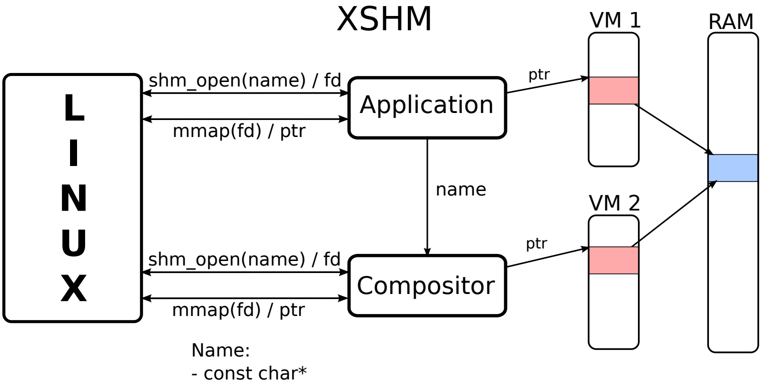
Connecting to the X-Server



2D application's rendering

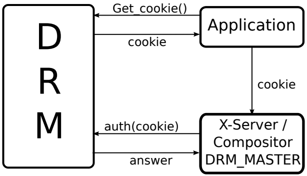


XSHM : X SHared Memory



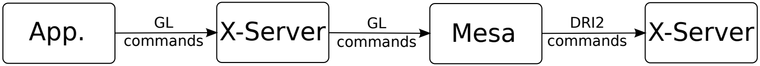
Rendering 3D applications

DRM Auth



3D application - direct rendering

3D application - indirect rendering



DRI-next

DRI-next : Fixing some of DRI2's deficiencies

- Drop the client authentification (render-node project);
- Let the applications allocate their buffers (Wayland-like):
- Use DMA-buf instead of GEM flink to share buffers.

Current status & ToDo

- Research and toying with unix sockets fd passing;
- See http://keithp.com/blogs/fd-passing/

Summary

Why should we care about the driver/hardware security?

Requirements

A driver/hw should not allow privilege escalation and should isolate GPU users:

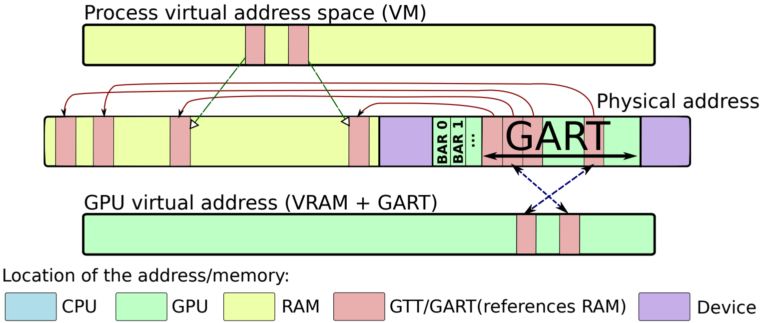
- Privilege separation: the driver/hw shouldn't give privileges;
- Confidentiality: read access to other buffers;
- Integrity: write access to other buffers.

Current status

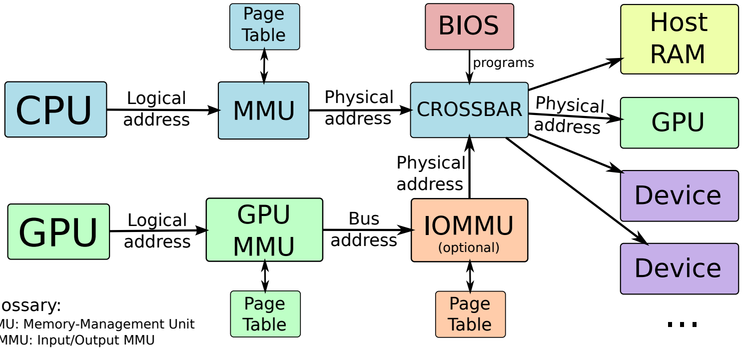
- Good access control to the RAM and VRAM from the CPU;
- The GPU may provide read-write-access to the whole VRAM/Host RAM range to UNIX users through the use of Shaders/GPGPU/copy-engines (TEGRA 2);
- The nVidia driver allows users to access the GPU's registers.

GTT/GART

Providing the GPU with easy access to the Host RAM



CPU & GPU Memory Requests Routing



Glossary:
MMU: Memory-Management Unit
IOMMU: Input/Output MMU
BIOS: Basic I/O System

Location of functions:

CPU Chipset GPU RAM BIOS Device

Driver/Hardware security : Current solutions

Expose a secure API to the userland

Goal: Users shouldn't be able to interfere with other GPU users

- The kernel should expose a sane API that isolate GPU users;
- This API should be the only way for a user to access the GPU;
- $\mapsto$ no regs should be accessible from the userspace!

Restrict GPU's RAM access rights

Goal: Deny access to the GPU to the kernel's internal structures or other programs' data.

- VGA window: The GPU can access the first 1.5MB of RAM;
- AGP aperture: Allow GPU access to a fixed part of the RAM;
- IOMMU: Programmable MMU for devices to grant RAM access as needeed where needed.

Driver/Hardware security : Current solutions

Isolate users in a separate VM

Goal: Restrict a GPU user to its own data by abstraction the memory address space

- Most secure solution;
- Increase context-switching delay (problem with DRI2 and Qt5)
- Currently used by: Nouveau (geforce 8+);
- Could also be used by: AMD (Southern Island+), Intel (Sandy Bridge+), ...

Driver/Hardware security : Current solutions

Isolate users through Command Submission validation

Goal: Restrict a GPU user to its own data by checking the commands issued by the user

- Lower context-switching delay;
- Higher CPU usage in kernel space;
- Currently used by: Radeon, Intel;
- Can be used by: any driver on any card.

Summary

Conclusion

Goals

- Make it possible to implement activities and provide secure isolation between them (like QubesOS/PIGA-OS);
- Allow the user to decide what he wants (per-application isolation vs performance?);
- Be ready for GPGPU shared clusters and the soon-to-come WebGL applications.

Current state

- No confidentiality/integrity between applications run by the same user:
  $\mapsto$The Linux graphics stack make it possible to spy on users.
- No input security on X11, Wayland can be secured
- DRM2 and DRI-next will improve security

Thank you for listening!

Martin Peres: martin.peres@labri.fr

Timothée Ravier: timothee.romain.ravier@gmail.com