Google

Defence in Depth: Sandboxing the UMD

Rob Clark
XDC2023

First, some terminology

- KMD - kernel mode driver (ie. drivers/gpu/drm)
- UMD - user mode driver, vk or gl (ie. mesa)
- Origin - web protocol + hostname + port
  - https://developer.mozilla.org/en-US/docs/Glossary/Origin
  - https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Hostile content vs vulkan/gl drivers /o\

- webgpu/webgl
- vk: specifically calls out Undefined Behavior:
  - “When an application supplies invalid input, according to the valid usages in the spec, the result is undefined behavior. In this state, Vulkan makes no guarantees as anything is possible with undefined behavior.”
- gl: not called out in spec
  - But alluded to in a few places

Hostile content vs vulkan/gl drivers /o\

- webgpu and webgl implementations provide some protections
  - Ex. bounds checking, some shader validation
- But still not so hard to crash a gl/vk driver if you try
  - And sometimes if you don’t try

What if we flipped it around?

- Instead of trying to guard against all “bad” usage..
  - And play never-ending-whack-a-mole

What if we flipped it around?

- Instead of trying to guard against all “bad” usage..
  - And play never-ending-whack-a-mole
- We consider the UMD as the “crumple zone”
  - Just plan on the vk/gl driver getting 0wned
  - And architect the system in a way that it doesn’t matter
- Not necessarily *removing* webgl/webgpu guards
  - Just plan on them not being sufficient
  - And design around that

How would that look?

- Per-origin, restrictive sandboxed process for UMD
- Conveniently also fits better with drm (KMD) security model
  - Separate GPU address space per origin
  - Information leak mitigations applied across origin boundaries
  - Etc
- Fits with web same-origin policy:

> “The same-origin policy is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin.
>
> It helps isolate potentially malicious documents, reducing possible attack vectors. For example, it prevents a malicious website on the Internet from running JS in a browser to read data from a third-party webmail service (which the user is signed into) or a company intranet (which is protected from direct access by the attacker by not having a public IP address) and relaying that data to the attacker.”

But… Sandbox process still needs access to KMD /o\

- Where there can also be bugs:
  - https://crbug.com/1436018 - refcount leak in msm_ioctl_gem_submit
  - https://crbug.com/1420161 - UAF in kfd_ioctl_unmap_memory_from_gpu
  - https://crbug.com/1415129 - Uninitialized Pointer in `msm_parse_post_deps`
  - https://crbug.com/1411997 - UAF in i915_gem_execbuffer2_ioctl
  - https://crbug.com/1406165 - UAF in i915_perf_add_config_ioctl
  - https://crbug.com/1400113 - UAF in panfrost_ioctl_create_bo
  - https://crbug.com/1401562 - UAF in drm_gem_object_release_handle3
  - https://crbug.com/1401560 - UAF in drm_gem_object_release_handle2
  - https://crbug.com/1393499 - UAF in drm_gem_object_release_handle
  - https://crbug.com/1293640 - i915 Linear Out-Of-Bound read and write access
  - … not a complete/representative list
- Bugs vs all-the-kernel-versions!
- We don’t want code execution to UMD to be a path to code execution in kernel!

Can we isolate UMD from KMD?

- We kinda already have this for VM guests: virtgpu native context!
  - Native UMD in guest (which we don’t trust, and can safely crash)
  - Native KMD in host
- virtgpu native context over a socket?
  - We kinda already have this too.. Vtest
  - (at least good enough for a prototype)
  - (and useful for testing)

This has some nice properties

- Already a security barrier in the VM use-case
- Single thread to deserialize KMD ioctl calls
  - This alone blocks 80% of the KMD security bugs
  - Blocking waits done on UMD side (ie. `poll(dmafence_fd)`, etc)
- Helps ensure ioctls are well formed
  - We know ptrs in ioctl payload are to valid memory/size
  - No weird things like ptr to unfaulted GEM objects
  - Sanitized handles
- virtio/vdrm helper - !24733
  - Shared code for native-context implementations
  - “For-free” vtest support[*]

Performance?

- It does make cmdstream submit ioctl synchronous
  - Ie. we have to wait for dma-fence to be created, no guest fences
- But round trip latency less than for VM guest
  - 10-15% hit for pedantic high-fps, single-draw cases
  - Not really measurable for more realistic workloads
  - Seems like a reasonable tradeoff for hostile environments

Other alternatives?

- One-way switch to enable per-`drm_file` serialization?
  - Would close off race condition exploits
  - But how to handle blocking ioctls?
  - Doesn't help with malformed ioctl issues
- Rust?
  - Definitely would help with untrusted input (ie. uapi) parts of driver
    - (I'd focus first an rendernode uapi)
  - It will take a long time to get there

Thank You