Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

Introduction

This white paper describes potential security vulnerabilities within an embedded system, specifically focusing on the Graphics Processing Unit (GPU). For the purposes of this white paper, security is defined as protecting the confidentiality of data within its domain; that is, preventing unauthorized access to confidential and classified data. The baseline for this discussion is that graphics applications run in a multi-partition or hypervisor environment with appropriate system level access protection through virtual address spaces and partitioning provided by the Real-Time Operating System (RTOS). A description of how GPUs are supported in multi-partition and hypervisor architectures is provided in How to Implement Multiple GPU Applications in an Embedded System. It is a given that applications accessing confidential and classified data run in their own separate partition(s). It is also assumed that any physical access protection like preventing hardware probing such as JTAG is provided at the hardware module or system level as necessary.

The concern is preventing unclassified applications from accessing classified data, either maliciously or accidentally. From a graphics view point, classified data includes graphics data used to render classified information to a display where a GPU may render both classified and unclassified data to one or more displays; for example, the system enables the execution of applications managing unclassified data alongside applications managing classified data.

GPU architectures are vulnerable to data leakage through a lack of memory isolation for GPU global memory, shared memory and register spaces. This stems from the fact that GPU hardware includes processing elements, DMA engines and that it has access to global system memory via a bus interface (for example, PCIe) as well as access to all video memory. Access to the GPU's hardware capability is controlled through a graphics driver such as OpenGL. For the remainder of this white paper, the focus will be on OpenGL drivers as they are the primary line of defense for securing rendered data.

There are three key areas of vulnerability to understand: the OpenGL driver pedigree, OpenGL driver Application Programming Interface (API), and addressable memory/registers.

Graphics/OpenGL Areas of Vulnerability

OpenGL Driver Pedigree

OpenGL driver pedigree, while mainly associated with the actual driver selected for implementation, may also involve subversion through attempts to bypass the driver and control the hardware directly.

Commercial OpenGL drivers from GPU vendors are based on a large number of lines of source code which may have been developed offshore and may contain third party and open source code. This may

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

leave vulnerabilities including designed-in back-door access either for intentional bypass or for testing and debugging purposes.

GPU registers and access mechanisms are available publicly as register information from the GPU vendor and as open source OpenGL drivers. This information reveals how to utilize the GPU hardware.

OpenGL Driver API

The OpenGL driver API also includes an industry standard EGL API for context management. For the purposes of this white paper, they are collectively referred to as "the OpenGL driver".

OpenGL consists of several APIs that move, copy and read back memory which can be used to gain access to memory containing confidential and classified data.

Addressable Memory/Registers

While multi-partition and hypervisor implementation includes RTOS memory access protection mechanisms including one or more, levels of address translation, there remain mechanisms to access memory and GPU registers.

GPU DMA buffers and driver memory space are addressable by the OpenGL driver. The GPU DMA buffers contain GPU rendering commands that could be copied and used to render the classified data from an unclassified application.

There is also the capability of the GPU processors to run shader programs which can access memory to move and copy data.

While GPU registers can be accessed, classified data is not stored there, nor would accessing registers be a direct path to accessing confidential and classified data. Furthermore, in some use case scenarios it may be possible to remove the GPU register space mapping from some partitions in the system entirely. In such scenarios, other safety critical and/or trusted partitions would be relied on for initialization of the GPU.

While there is Non-Volatile memory required by a number of GPU implementations (the VBIOS), the OpenGL driver does not provide an API to write/read to/from this storage device. Therefore, classified data would not be stored there intentionally or as part of a graphics application. With public GPU register information, it is possible for a malicious application to access the VBIOS. The hardware could be designed to implement a write protect feature on the VBIOS such that it could not be written to when deployed.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

Graphics/OpenGL Considerations for Multi-Level Security

While CoreAVI has delivered OpenGL drivers for use with Multi-Level Security (MLS) enabled operating systems, there are additional OpenGL driver features available to further enhance security in a MLS environment. These features are provided through the SecureCore™ option for ArgusCore SC™.

The OpenGL Driver Pedigree

To ensure OpenGL pedigree, the OpenGL driver should be sourced from a trustworthy partner and access to source code for security inspection should be available.

When it comes to knowledge gained from open source drivers to directly access the GPU, the GPU access is protected by the GPU virtualization manager and with a single supplier responsible for this and the GPU virtualization manager aware OpenGL driver in the partitions, the access to the GPU remains obfuscated as much as the supplier does not disclose the communication solution.

OpenGL Driver API

For the OpenGL APIs that read back, move and copy memory, which are required for most applications (for example, to get texture data to the framebuffer for display), these APIs require a handle to a surface. By adding a mechanism to the OpenGL driver to associate a partition identification with the rendering surface handle, access to rendering surfaces can be restricted to partitions authorized to do so.

Addressable Memory/Registers

Modern embedded GPUs like the AMD Radeon™ E8860 include hardware based virtual memory management. With GPU hardware based virtual memory management, the OpenGL driver can set-up each OpenGL context with its own GPU virtual memory context. Any attempt by the GPU (or DMA engines therein) to access memory outside of the memory ranges the driver has allocated to that OpenGL context would result in an GPU memory controller fault and an error being raised. This ensures that OpenGL contexts are limited to the memory space they can access.

The OpenGL driver can allocate the DMA buffer region used by the driver in RAM outside of any Virtual Machine region so that if an IOMMU is available, e.g. Peripheral Access Management Unit (PAMU), it can be configured to cover this region without exposing Virtual Machine memory space. This helps ensure that when only coarse IOMMU region sizes are available from the target Single Board Computer (SBC) hardware it can be possible to provide further assurance on the restricted address space access of the DMA engines of the GPU such that they cannot read/write from/to Virtual Machine memory spaces when an appropriate IOMMU configuration is applied.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

Only the portion of the GPU's DMA buffer which will be used exclusively by a partition is mapped into that partition's address space. Thus, the CPU's MMU will protect each partitions DMA buffers from accidental or malicious pointer access from another partition.

The video memory (VRAM) access aperture should be for the exclusive use of video decode (DecodeCore™) and only be mapped to partitions requiring the use of DecodeCore. This prevents the potential to use the aperture to maliciously read back the contents of video memory which may result in reading confidential or classified information. Therefore, all partitions using DecodeCore must be at the same security level. This vulnerability may be addressed with newer GPU hardware generations.

With OpenGL SC 1.0.1 there is no access to load and run shader programs so no vulnerability resulting from an unclassified program loading a shader to read or copy system or video memory containing confidential and classified data. All shaders executed by the GPU execute within the virtual memory space of their OpenGL context, which means that even if an application constructed a malicious shader attempted to read/write memory outside of the range it has been allocated, the GPU's memory controller would fault while executing this shader and an error would be raised.

Another potential method to access confidential and classified data is when memory that was used for confidential and classified data is un-allocated and re-allocated to an unclassified application which could then read the previous data. Normally this would require a memory scrub to be associated with a memory free before placing the memory back into the pool, however, it is recommended to use Safety Critical OpenGL drivers which do not free and re-allocate memory which is part of the safety requirements. Memory is allocated during initialization and remains with the respective applications throughout.

In addition, OpenGL drivers should only allow the partition that initialized the driver to un-initialize it. This, coupled with a memory clear of VRAM during initialization, will remove the potential for a malicious partition to un-initialize and then re-initialize the driver and attempt to create surfaces to read previous data from VRAM which would otherwise still contain confidential or classified data.

Core Avionics & Industrial Inc.
400 North Tampa Street
Suite 2850
Tampa, Florida 33602

T: 888-330-5376
F: 866-485-3199
www.coreavi.com

Where to Find Additional Information on OpenGL Drivers with MLS Support

CoreAVI supports Multi-Level Security for OpenGL drivers as described in this white paper. CoreAVI develops all its drivers from the ground up in North America. These drivers do not contain any third party or open source software. Therefore, the drivers themselves do not contain hidden backdoor access vulnerabilities. CoreAVI can make the driver source available at its site for security audits as needed. A license for the driver source code is also available enabling rebuilding and testing of the driver.

The GPU Manager contained in CoreAVI's HyperCore™ product is a common communication channel between all partitions and is designed in such a way that it does not copy data buffers from partitions. The GPU Manager does not provide information back to partitions other than success or failure of their request and therefore it is not possible for it to leak information from one partition to another. HyperCore is essentially a routing agent passing pointers to data between partitions and the GPU.

CoreAVI can support National Information Assurance Partnership (NIAP) Common Criteria validation at the application level through a support services agreement.

CoreAVI provides high Technology Readiness Level (TRL) OpenGL SC 1.0.1 and OpenGL SC 2.0 drivers with a SecureCore option and HyperCore GPU virtualization manager with optional certification evidence for avionics, automotive and railway safety critical graphics functions.

More information about CoreAVI's ArgusCore OpenGL drivers, with extensions, along with a comparison of driver features can found here: CoreAVI safety certifiable graphics drivers.

More information about CoreAVI's HyperCore GPU Virtualization Manager can be found here: CoreAVI HyperCore

Contact CoreAVI to find out what we are working on and to discuss your demonstration/evaluation requirements: Sales@CoreAVI.com