Understanding The Security of Discrete GPUs

Zhiting Zhu¹, Sangman Kim¹, Yuri Rozhanski², Yige Hu¹, Emmett Witchel¹, Mark Silberstein²

1. The University of Texas at Austin  2. Technion-Israel Institute of Technology
Outline

● IOMMU mitigation
● GPU microcode attack (if time permits)
GPU as a host for stealthy malware

1. Threat Model
2. IOMMU attack
3. GPU microcode attack
Threat model

Attacker:

- Load and unload kernel modules via module loading capability.
- Access the GPU control interface i.e., MMIO register regions.
- Loses the module loading capability and is allowed only unprivileged access after the malware is installed.

Stealthiness

- Originate with the GPU reading and writing CPU memory.
DMA attack

- GPU is a programmable device.
- Easier to launch DMA attack compared to other DMA capable devices.
- GPU microcode attack.

![Diagram showing DMA attack between IO Device, GPU, and Memory. The diagram illustrates the flow of DMA request and the relationship between device and memory addresses.](image-url)
IOMMU

- Hardware
- Software management
- IOMMU attack
IOMMU

- Maps device addresses to CPU physical addresses.
- Check access permission.

IO Device

IOMMU

IOTLB

Memory

Kernel data structure
- Not kept coherent with the IO page table by hardware.
- Software must explicitly flush the cached mappings when they are removed from the IO page table.
# IOMMU configurations

<table>
<thead>
<tr>
<th>Mode</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>● Default configuration for many Linux distributions.</td>
</tr>
<tr>
<td></td>
<td>● Reduce IO performance.</td>
</tr>
<tr>
<td></td>
<td>● Incompatible with certain devices and features.</td>
</tr>
<tr>
<td>Pass through</td>
<td>● Hardware IOMMU is turned off.</td>
</tr>
<tr>
<td></td>
<td>● Device address is used as CPU physical address.</td>
</tr>
<tr>
<td>Deferred</td>
<td>Default mode when IOMMU enabled.</td>
</tr>
<tr>
<td>Strict</td>
<td>IOMMU enabled.</td>
</tr>
</tbody>
</table>

---

**Security**

- **Not secure**
- **Secure**

**Performance**

- **Fast**
- **Slow**
# IOMMU configurations

<table>
<thead>
<tr>
<th>Mode</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>● Default configuration for many Linux distributions.</td>
</tr>
<tr>
<td></td>
<td>● Reduce IO performance.</td>
</tr>
<tr>
<td></td>
<td>● Incompatible with certain devices and features.</td>
</tr>
<tr>
<td>Pass through</td>
<td>● Hardware IOMMU is turned off.</td>
</tr>
<tr>
<td></td>
<td>● Device address is used as CPU physical address.</td>
</tr>
<tr>
<td>Deferred</td>
<td>Default mode when IOMMU enabled.</td>
</tr>
<tr>
<td>Strict</td>
<td>IOMMU enabled.</td>
</tr>
</tbody>
</table>
IOMMU configurations

<table>
<thead>
<tr>
<th>Mode</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>● Default configuration for many Linux distributions.</td>
</tr>
<tr>
<td></td>
<td>● Reduce IO performance.</td>
</tr>
<tr>
<td></td>
<td>● Incompatible with certain devices and features.</td>
</tr>
<tr>
<td>Pass through</td>
<td>● Hardware IOMMU is turned off.</td>
</tr>
<tr>
<td></td>
<td>● Device address is used as CPU physical address.</td>
</tr>
<tr>
<td>Deferred</td>
<td>Default mode when IOMMU enabled.</td>
</tr>
<tr>
<td>Strict</td>
<td>IOMMU enabled.</td>
</tr>
</tbody>
</table>
When system memory is unmapped from IO devices:

Clear the entry in IO page table
When system memory is unmapped from IO devices:

Clear the entry in IO page table

<table>
<thead>
<tr>
<th>IOTLB Flush</th>
<th>Deferred Mode</th>
<th>Strict Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strategy</strong></td>
<td>Flush entire IOTLB.</td>
<td>Flush individual entry in given domain.</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>When deferred list is full or 10 ms after the first entry, whichever comes first.</td>
<td>Immediately after unmapping entry from IO page table.</td>
</tr>
</tbody>
</table>
When system memory is unmapped from IO devices:

Clear the entry in IO page table

<table>
<thead>
<tr>
<th>IOTLB Flush</th>
<th>Deferred Mode</th>
<th>Strict Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strategy</strong></td>
<td>Flush entire IOTLB.</td>
<td>Flush individual entry in given domain.</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>When deferred list is full or 10 ms after the first entry, whichever comes first.</td>
<td>Immediately after unmapping entry from IO page table.</td>
</tr>
</tbody>
</table>
IOMMU attack

1. Writes a malicious IO page table entry.
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.
3. Overwrite the IO page table.
How long can a stale entry last in IOTLB?

<table>
<thead>
<tr>
<th>Workload</th>
<th>Bit rate</th>
<th>Stale period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle ssh connection</td>
<td>10 bps</td>
<td>1 day</td>
</tr>
<tr>
<td>Web radio</td>
<td>130 Kbps</td>
<td>1 hour</td>
</tr>
<tr>
<td>Web video: Auto (480p)</td>
<td>2 Mbps</td>
<td>1 min</td>
</tr>
</tbody>
</table>
Stealthiness

- IOTLB entry is not accessible by software.
- IO page table can be monitored by security tools.
Conclusion

- Discrete GPUs are not an appropriate choice for a secure coprocessor.
- Discrete GPUs pose a security threat to computing platform.
IOMMU attack

- Defeat strict mode.
- Won’t work in deferred mode since it flush entire IOTLB.
GPU as a host for stealthy malware

1. Threat Model
2. IOMMU attack
3. GPU microcode attack
GPU microprocessor and micro code attack

Can access GPU and CPU memory

Video display

Power management

Decryption

GPU Chipset

CPU

GPU

Process

Kernel

Chipset

CPU RAM

GPU RAM

PCIe Bus
GPU microcode attack

1. Loads the attack code into microprocessor storage using MMIO registers (details in the paper).
GPU microcode attack

1. Loads the attack code into microprocessor storage using MMIO registers (details on paper).
2. The microcode transfers data from GPU memory to its own memory.
1. Loads the attack code into microprocessor storage using MMIO registers (details on paper).
2. The microcode transfers data from GPU memory to its own memory.
3. Launch the attack by writing to critical data structures in CPU memory.
Microcode validation

- Microcode needs to be signed by NVIDIA.
- No code modification after initialization.