Honeycomb: Secure and Efficient GPU Executions via Static Validation

Haohui Mai, Jiacheng Zhao, Hongren Zheng, Yiyang Zhao, Zibin Liu
Mingyu Gao, Cong Wang, Huimin Cui, Xiaobing Feng, Christos Kozyrakis
AI on private data needs S&P solutions

An astronaut riding a horse

Ban 3rd-party AI services on private / internal data
GPU TEE: a pragmatic approach

- Trusted Execution Environments (TEEs): isolation among apps and the host
  - 😃 Efficient: native speeds within enclaves
  - 😃 Ease of adoption: using models as-is

- Current proposals
  - 😞 Hardware modification: slow evolutions
  - 😞 Driver-based: large TCB (>1M SLOC), real bugs in security features, undermining trustworthiness
Honeycomb: confining behaviors via static validation

- Software-based GPU TEE
  - Isolation: static analysis over GPU binaries
  - Minimize TCB: OS-level security monitors (SM)
  - Efficient IPC: co-designing static analysis & OS primitives
- **Flexible, efficient and secure**
  - Complement hardware limitations
  - Security checks at load time: 2% overheads for BERT / NanoGPT. Modest overall dev. efforts
  - System-wide invariants: 18x smaller TCB compared to Linux, 529x faster for secure IPC
Agenda

- Introduction
- Assumptions & background
- Design & implementation
- Evaluation & experiences
- Conclusion
Threat models

- Goal: enforce confidentiality and integrity for *multiple mutually distrusted* applications

- Protection for each application + isolation from the untrusted host environment

- Adversary
  - Controls entire software stacks (OS / compiler / hypervisor)
  - Has physical access of the hardware
  - Sniffs PCIe traffic
  - Cannot tamper the CPU or GPU silicons

- Assumptions
  - CPU TEE (e.g., AMD SEV-SNP)
  - Discrete GPU with integrated memory
  - Trusted I/O paths: HW (e.g., AMD SEV-TIO) or SW approach (detailed in the paper)
  - Side-channel attacks are out of scope
GPU is a discrete accelerator

- Operations $\implies$ host cmd. queues (mapped in GPU)
- GPU executes kernel functions (in GPU mem.)
- Exchange data over the PCIe bus (untrusted)
- GPU drivers
  - Initialize hardware and addr. spaces
  - Multiplex device memory / queues
Agenda

• Introduction

• Assumptions & background

• Design & implementation

• Evaluation & experiences

• Conclusion
Overview of Honeycomb

- Regulating host-GPU interactions
  - SVSM (Secure VM Service Module) in SEV-SNP VM: cmd. queue (App)
  - SM (Security Monitor) in the sandbox VM: GPU driver
  - CPU TEE to enforce integrity of SVSM/SM
- Validator inside SVSM ensures that all GPU kernels from applications are safe
- System-wide invariants $\Rightarrow$ efficient IPC
Security monitors: intercepting at lowest levels

- Remove OS kernel / GPU driver from the TCB
- Init sequences, memory isolation, validated kernels, secure memcpy, …

```c
void check_launch_kernel(
    AddrSpace *addr,
    DispatchPkt *p) {
    if (!validated(addr, p->kernel_object))
        abort_user();
    ...
}
```

- Challenge: recover high level semantics from low level operations
  - Two 32-bit MMIO writes $\Rightarrow$ address of a buffer

- Found 5 new bugs in AMDGPU driver, fixes deployed in Linux 5.19
Static validation: analyzing binaries

- Remove compilers from the TCB
- Well-formed reads, control flow integrity
- Memory access: partitioned virtual addr. space
- Analyze the range of each memory access
- No writes to undermine the integrity

```c
void fill(int *base, int n, int b) {
  u32 dim = blockDim.x;
  u32 gid = blockIdx.x;
  u32 lid = threadIdx.x;
  u32 tid = gid * dim + lid;
  if (tid < n)
    base[tid] = b;
}
```
Polyhedral analysis + GEMM = ✌️

- Range checks using polyhedral analysis: techniques from auto parallelization
- Minimal overheads for ML workloads
- Flow-sensitive, path-insensitive algorithm
  - Simple algorithm to minimize TCB
- Complex programs: add runtime checks
  - e.g., indirect heap references
- Implementation challenge: deal with complexities of analyzing binaries

void fill(int *base, int n, int b) {
    u32 dim = blockDim.x;
    u32 gid = blockIdx.x;
    u32 lid = threadIdx.x;
    u32 tid = gid * dim + lid;
    if (tid < n)
        base[tid] = b;
}
Efficient IPC: secure *direct memcpy*

- Useful primitives for multi-stage pipelines
- Validation enforces protected IPC regions in address spaces
  - Avoid double en-/decryption
- More details in the paper
Agenda

- Introduction
- Assumptions & background
- Design & implementation
- Evaluation & experiences
- Conclusion
Security monitors minimize TCB

- 2 EPYC 7433 CPUs, 1 AMD RX6900XT GPU
- Linux 5.17, ROCm 5.4.0
- TCB of Honeycomb (~82 KLOC): 18x smaller
  - Linux kernel ~1.5MLOC
  - Core functionalities
  - Drivers (AMDGPU) and libraries (DRM & TTM)
  - Userspace drivers (ROCm) ~400 KLOC
Static validation is efficient

- 5 benchmark suites, HPC, CV, ML (DNN/Transformer). 23 apps in total
- Relative end-to-end execution time from 0.71-1.31 compared to the Linux stack
- Simpler drivers, breakdowns in the papers
- Efficient on ML workloads
  - 2% overheads for BERT / NanoGPT
- Spent most time on GEMM kernels: polyhedral analysis works well
- Modest dev. effort to pass validations
Conclusion

• Honeycomb supports secure and efficient GPU executions

• Static analysis (validation) is a practical and flexible technique for GPU apps.
  • Honeycomb enhances security via co-designing validation + OS support
  • Efficient on real-world workloads

• The end-to-end SW/HW stack for GPU evolves quickly
  • A promising technique to explore novel designs
Thank you!