CSE290X Final Report: Drywall: Reinforce the CXL malicious state from kernel and TDX applications

Yiwei Yang
University of California, Santa Cruz
Santa Cruz, California
yyang363@ucsc.edu

Yangyu Chen
Chongqing University
Chongqing, China
cyyself@cqu.edu.cn

ABSTRACT

The workloads of Meta's Metaverse, along with other in-memory databases, indicate a high demand for memory capacity with acceptable memory bandwidth and latency. To meet this demand, the emerging CXL device standard offers a new type of byte-addressable remote memory with various memory types and hierarchies, which is well-suited for the demand. Additionally, the CXL3.0 device allows for multi-nodes' access to multi-memory expanders semantics, expanding the span set to a large shared memory pool beyond the capabilities of current operating systems. However, there are security concerns with the CXL protocol, where cacheline exclusiveness in the Linux kernel or programs running inside the enclave may be vulnerable to physical attack bugs within the CXL protocol. To address these concerns, this project proposes testing for vulnerabilities within the Linux kernel bare metal and TDX program, and reinforcing the IOMMU mechanism to prevent data leakage and system crashes.

1 INTRODUCTION

The topic of timing side-channel attacks and physical shutdown of a CXL device to attack the kernel location is intriguing because there are already many attacks on PCIe devices, such as timing side-channel attacks based on timing differences and kernel fuzzing mitigation in user-mode, and thunderbolt-attached devices that can kill kernel space processes. Other protection measures like PCIe DMA cache line level firewall protection can filter DMA requests through PCIe to prevent attackers from making malicious moves. A hardware-software codesign methodology on RISC-V microcode can mask data that is not accessed by the attacker's side channel physical attack. However, if we accidentally take the CXL.cache devices offline while they are in possession of the cache line, it could be risky if a malicious cacheline is received from a dead or maliciously built device while the cache line is in the kernel. To mitigate this risk, sensitive data in the kernel or devices behind the TDX should be moved every time the Type 1 device takes an exclusive cacheline. Additionally, if the program behind the TDX is a verified database, we may need to give deferred verified results for all CXL endpoints, which involves moving the previous memory node computability to the CXL controller. DMA is also not trusted since it's on-demand without leaking information, so we can utilize the computational power in the CXL controller to achieve similar protection. You can see the code in linux, Drywall and qemu-image

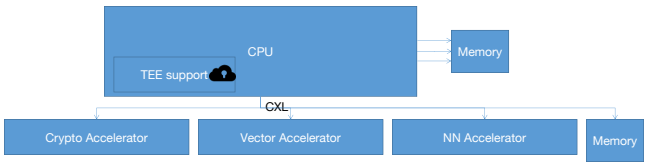
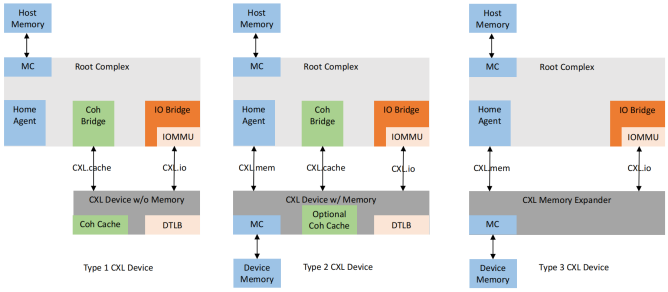


Figure 1: CXL topology

1.1 CXL Basics



Extending CXL to outer devices divides the solution into 3 parts, CXL.io meaning the original PCIe operations, CXL.cache meaning the outer devices can manage the cacheline inside the CPU and even share the device DRAM as LLC, this provides great computability of where the device defined when and how the data is used, all the CXL attached devices will aware of all the cacheline changes assured by the protocol, CXL.mem meaning the CPU can access through the serial bus through PCIe and issue ld/st instruction to devices. and same as memory, the CXL managed memory can cache some of the memory windows that next time access if the data in the window is not dirty, there's no need for extra interaction with the CXL root complex.

Type 1 devices can utilize CXL.io or CXL.cache to control the control plane of PCIe devices, and use CXL.cache to issue data plane operations. One such unique device is a crypto accelerator that shares some cachelines with the program in the CPU. Type 2 devices offer all three functionalities, and a unique example is a GPU. Type 3 devices provide CXL.mem and CXL.io functionality, meaning they are essentially memory devices.

2 BACKGROUND

The recent CXL technology brings up an excellent opportunity for accelerators and memory disaggregations. In the past, we make everything in the SoC like Apple, AMD Zen, or Qualcomm. They put the accelerator inside the core so that all are connected to the internal memory bus. The coherency information is shared across the devices and TEE support inside the memory controller can

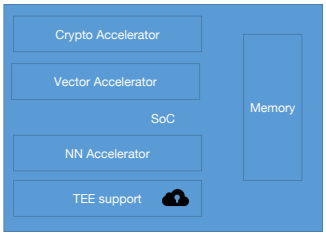


Figure 2: SoC topology

make sure all the access to these accelerators is safe. Since SoCs are packaged together, the main bus connected together is hard to be broken. Even though, we have a great number of vulnerabilities described in **??**.

However, with the chip design's limit comprehensively considering the die size physical limit, memory bandwidth limit, and heat dissipation, Apple and other companies've been stuck with the current num of cores and accelerators. Intel leads the CXL protocol that gives every outer device ability to define the CPU's cacheline. We can plug memory and maintain a TEE support for other type3 devices' memory expander with PCIe serial lane. This kind of design helps a variety of accelerators(type 1 devices) share the cacheline coherency. Here introduce a lot of problems of whether taking exclusive-able. For now, we can simply utilize the bit of ATS extended by CXL specification[5] for mitigating the bug, but in the long run, we consider putting an eBPF co-processor near slower CXL devices to co-design the memory access and support full security of the main bus.

3 THREAT MODEL

(1) Lower ability caused by the device hot-plug or stop functioning by hardware or software bugs
  (a) The PCIE(CXL.io) has no persistent effect when the device is not safely removed.
  (b) For the CXL.cache device, if some cache line being modified state resides in this device, and the cache line cannot be safely recovered from the dead device, which causes data loss and even kernel panic or system shutdown.
(2) Lower security caused by new types of side channels.[10]
(3) Although the memory bus is enhanced for future multi-processor systems, sync between each device will have much more memory access latency compared to Today's CPU hierarchy.
  (a) For example, a shared queue or a memory region is used to reduce operations.

We think the exclusive cache bugs extension to the outer devices is lethal to the kernel state, but the assumption we are proposing is too strong, that we need buggy CXL Type1 devices, and the sensitive data will be offloaded to this device and if for 3a, we require the NIC to be SRIOV(a shared queue inside the NIC to make multi VMs share) and has bugs. We here just want to implement a type 1 crypto device in the kernel and the threat model is the mal state or unplugging of the crypto accelerator will not hurt the runtime of the file system that relies on the accelerator, we will mitigate it by labeling every cacheline request with ATS to know it's taken

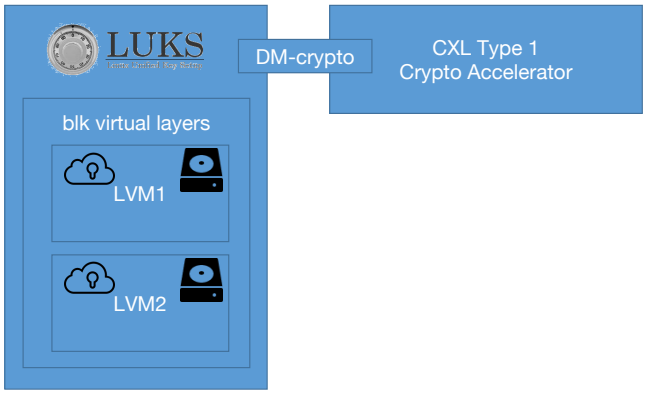


Figure 3: LUKS topology

exclusiveness-able and will transfer the calculation to kernel's software implementation if we detect the accelerator is in mal state or unplugged.

4 PROPOSED DESIGN

4.1 A CXL Type 1 device simulator in the qemu

We first need to simulate the type 1 crypto accelerator in the qemu. We will reuse the code of dm-crypt[6] and add a subpage[8] snooper for all the access to the qemu managed devices' cacheline and add CXL transportation over that we can put our ATS on. This is a hardware-software co-design method that in the runtime, the crypto algorithm needs to tell which part of the cacheline is taken exclusively so that the device can access the cacheline, next time you write to the cacheline you need cache another replica data in the Private DRAM of the cacheline state, if the device is unplugged, we fall back to the software implementation of the crypto that is in the slower path can recover and resume from the state the device previously stored in the private DRAM.

5 PROPOSED EVALUATION

Our proposed evaluation will be first implemented on the QEMU for PoC and then for future submission to the conference and wait until the physical devices are coming we will implement real-world recovery resolutions.

5.1 LUKS

The LUKS[4] is a key setup for one specified storage device, and every access to the device will use the key stored in the TPM to encrypt and decrypt, the crypto calculation middle state may hurt the data safety.

5.2 QEMU implementation

Initially, we implement the basic type-1 CXL.cache operations for naive CXL devices. After examining the current mechanism within the qemu and kernel, we will construct a state that may adversely impact the IO stack if we offload the LUKS crypto part to the crypto accelerator. To mitigate this issue, we will employ a hardware-software co-design approach to prevent sensitive state crashes.

Our design will involve attacking the original un-modified CXL devices to identify any additional vulnerabilities that the CXL.cache may introduce. We will then propose our ATS (Address Translation Services) bit mitigation and a perfect solution to fall back to a slower path when the CXL device is in a malicious state.

5.3 Implantation of type-1 CXL.cache device in the qemu

EPT-Based Sub-Page writes Protection (SPP) provides Virtual Machine Monitor (VMM) with the ability to specify write permissions for guest physical memory at a sub-page granularity of 128 bytes. With SPP, the hardware enforces write-access checks for sub-pages within a protected 4KB page, providing fine-grained memory protection for use cases such as memory guard and VM introspection.

To activate SPP, the "sub-page write protection" bit (bit 23) must be set to 1 in Secondary VM-Execution Controls. The feature is backed by a Sub-Page Permission Table (SPPT), with the subpage permission vector stored in the leaf entry of the SPPT. The root page is referenced via a Sub-Page Permission Table Pointer (SPPTP) in VMCS.

To enable SPP for guest memory, the guest page must first be mapped to a 4KB EPT (Extended Page Table) entry. Once mapped, the SPP bit (bit 61) of the corresponding entry must be set. During EPT walks, the hardware traverses the SPPT with the guest's physical address to look up the sub-page permission vector within the SPPT leaf entry. If the corresponding bit is set, writing to the sub-page is permitted. Otherwise, an SPP-induced EPT violation is generated.

In summary, SPP provides fine-grained memory protection for virtual machines, enabling VMM to specify write permissions for guest physical memory at a sub-page granularity. This feature is activated by setting the "sub-page write protection" bit in Secondary VM-Execution Controls and backed by a Sub-Page Permission Table. Once activated, SPP allows hardware to enforce write-access checks for sub-pages within a protected 4KB page, providing enhanced security for memory guard and VM introspection use cases.

5.4 Cache coherency emulation

```
struct D2HDataReq = {
    D2H DataHeader 24b;
    opcode 4b;
    CXL.cache Channel Crediting;
}
struct CXLCache = {
    64Byte Data;
    MESI 4 bit;
    ATS 64 bit;
    [D2HDataReq;n] remaining bit;
}
```

With Intel SPP write protection support for metadata, access to arbitrary cachelines can be marked with SPP. Transaction descriptions and queuing can be performed in the 64-byte residue data in the SPP, with the arbitrary operation based on the queue casting an effect on MESI bit change, update writes protection for the Sub Page and Root Complex, or other side effects like switch changes. Requests from the host and device are not scheduled in a First-In-First-Out (FIFO) manner. Instead, the host's data will have a better priority for D2H FIFO, and H2D requests will be consumed first. All operations follow the interface operation to CXLCache.

To enable taking exclusiveness-able, we mark the Transportation ATS bit to indicate whether exclusiveness can be taken. We then copy the cacheline to another map, and once emulated unplugged, the cacheline is copied back for further operation of the kernel to resume software crypto calculation. Regarding the emulation of eviction, we have two proposals. The first involves using qemu pebs to watch the cache evict of the physical address of an SPP. The second proposal is to use sub-page pin page_get_fast to pin to a physical address within the last-level cache.

5.5 Emulate Error

The CxlUncorErrorType (Enum) specifies the type of uncorrectable CXL error to inject. These errors are reported through an AER (Advanced Error Reporting) uncorrectable internal error, with additional information logged at the CXL device. We use this to inject runtime errors and reproduce the bugs that happen in the driver. After injecting the error during every stage of interacting with the cxl virtio crypto accelerator, we've been able to debug and fuzz the driver bugs.

6 REAL DEVICE IMPLEMENTATION

Our plan involves fully implementing an eBPF co-processor near CXL devices to address profiling and security concerns. By filtering all cacheline requests at the CXL bus level, we can improve request filtering to prevent kernel vulnerability.

6.1 Preliminary Idea for verified DB CXL memory expander over TDX

When it comes to verifying data in distributed systems, one approach is to use a merkle tree to store data and hash values. The concerto[1] algorithm takes verified steps on a single tree, which makes it an efficient solution for verifying data in a distributed environment. However, the concerto algorithm may not be suitable for verifying data on remote devices since it assumes a centralized system.

To overcome this limitation, [7] proposed a new approach where the verified process is distributed using RDMA (Remote Direct Memory Access). With RDMA, the remote devices' enclave can be leveraged to implement the verified process in a distributed manner. In this approach, a Merkle hash table is still used, but the verified process is offloaded onto the device on the eBPF (extended Berkeley Packet Filter) processor. The local enclave is then used to enlarge the verified memory by 1TB TDX (Total Memory Encryption) per key.

The use of eBPF co-processors situated near CXL devices can be leveraged for implementing this approach in real devices. The server algorithm can be placed on the server, and the client algorithm can be placed on each eBPF processor. This allows for an efficient implementation of the verified process on distributed systems, ensuring data integrity and security.

One advantage of this approach is that it can ensure the security of data in transit. Since the verified process is distributed using

RDMA, it allows for secure data transfer between devices. Additionally, the use of TDX can prevent data from being compromised in the event of a physical attack.

Overall, the proposed approach offers a more efficient and secure way of verifying data in distributed systems. By offloading the verified process onto the eBPF processor and leveraging the local enclave, the approach ensures that data is verified in a distributed manner, allowing for secure data transfer between devices.

7 RELATED WORK

We describe the two lines of this work from which Drywall takes inspiration:

The following points were discussed:

(1) Recent research by [9] revealed that a user-space driver issue can trigger kernel space shuts, and the author designed fuzzing techniques for real-world PCIe devices' malicious patterns to better identify kernel bugs. They found that SoC-attached devices with IOMMU have some malicious state. Another study uses a PCIe firewall to protect every DMA access to PCIe devices, with the hook filtering DMA controller traffic. Side-channel attacks pose a significant vulnerability to current cloud FPGAs [3], which could worsen if everything is extended with cache coherence manner [10]. Moreover, the ioctl inside the Linux kernel for PCIe-attached devices such as FPGAs has a timing difference for side-channel attack evaluation.

(2) [1, 2, 7] introduced Concerto, a verified key-value store that sends verified memory access from the enclave to the untrusted host, which stores a Merkle hash tree. They also apply deferred verification that verifies integrity after a million instructions were executed on the tree operation.

REFERENCES

[1] A. Arasu, K. Eguro, R. Kaushik, D. Kossmann, P. Meng, V. Pandey, and R. Ramamurthy. Concerto: A high concurrency key-value store with integrity. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 251–266, 2017.

[2] A. Arasu, B. Chandramouli, J. Gehrke, E. Ghosh, D. Kossmann, J. Protzenko, R. Ramamurthy, T. Ramananandro, A. Rastogi, S. Setty, et al. Fastver: making data integrity a commodity. In *Proceedings of the 2021 International Conference on Management of Data*, pages 89–101, 2021.

[3] I. Giechaskiel, S. Tian, and J. Szefer. Cross-vm covert-and side-channel attacks in cloud fpgas. *ACM Transactions on Reconfigurable Technology and Systems*, 16 (1):1–29, 2022.

[4] R. Hat. Configuring luks: Linux unified key setup. URL https://www.redhat.com/sysadmin/disk-encryption-luks.

[5] Intel. Cxl specification, 2022.

[6] A. Linux. Qemu qmp website. URL https://wiki.archlinux.org/title/dm-crypt.

[7] I. Messadi, S. Neumann, N. Weichbrodt, L. Almstedt, M. Mahhouk, and R. Kapitza. Precursor: A fast, client-centric and trusted key-value store using rdma and intel sgx. In *Proceedings of the 22nd International Middleware Conference*, pages 1–13, 2021.

[8] Y. Ozawa and T. Shinagawa. Exploiting sub-page write protection for vm live migration. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 484–490. IEEE, 2021.

[9] S. Qin, F. Hu, B. Zhao, T. Yin, and C. Zhang. Kextfuzz: Macos iommu firewall inspection fuzzing. *USENIX Security 23*, 16(1):1–29, 2022.

[10] F. Yao, M. Doroslovacki, and G. Venkataramani. Are coherence protocol states vulnerable to information leakage? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 168–179. IEEE, 2018.