# CSE290X Project Progress 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

Meta's Metaverse workloads together with many other in-memory database workloads tell us that the capacity of the memory is in great demand if given acceptable memory bandwidth and memory latency. The emerging CXL type 1 or type 2 device standard provides a new type of byte-addressable remote memory with a variety of memory types and hierarchies which is the best fit for the demand. Moreover, the CXL3.0 device unlocks the multi-nodes' access to multi-memory expanders semantics which enlarges the span set to a large shared memory pool of how current operating systems can address. However, the security issues that happen in the CXL protocol, that the taking cacheline exclusiveness inside the Linux kernel or programs that are running inside the enclave like TDX, will also be vulnerable to such physical attack "bugs" inside the CXL protocol. In this project, we propose attacks inside the Linux kernel bare metal and inside the TDX program to see there's vulnerability and reinforce the IOMMU mechanism to make sure the important data do not leak and the system does not crash. In future work, we plan to transplant concerto[1] to a programmable eBPF coprocessor aside CXL controller on type 1 or type 3 devices. We will discuss our extending verified kvs over the TDX plan for multiple servers and multiple clients on these CXL3.0-connected machines and memory pools.

## 1 INTRODUCTION

We found the topic of timing side-channel on CXL or physical shutdown of one CXL device to attack the kernel location interesting because PCIe devices already have many attacks like timing side-channel attacks based on timing differences in ioctl [10] to the different topology of the devices, and mitigation of fuzzing kernel [9] that in user-mode, thunderbolt attached devices can kill the kernel space stuff. Other protections like PCIe DMA cache line level firewall protection may filter all the DMA request through PCIe to get rid of the attacker's malicious move. [3] provides a hardware-software codesign methodology on RISC-V microcode to mask the data not being accessed by the attacker's side channel physical attack. Since the CXL.cache capability allows for the ownership of cache lines within the CPU and they're caching there, devices can do so. If we unintentionally take the CXL.cache devices offline or shut them down while they are in possession of the cache line, it is extremely risky. The following time, we received a malicious cacheline from a dead device or a maliciously built device if the cache line happened to be in the kernel. We decided to make sensitive data in the kernel or devices behind the TDX, every time the Type 1 device takes exclusive cacheline, we request the DMA the data to the private DRAM, and every time the type 1 device is shut, we recover the data to the kernel. If the program behind the TDX is verified DB, we may need to give deferred verified results for all CXL endpoints1, meaning we move the previous memory node computability to the CXL controller. Since DMA is also not trusted since it's on-demand without leaking information, we can leverage the computational power in the CXL controller to achieve similar things.

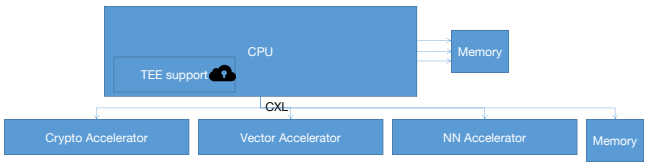
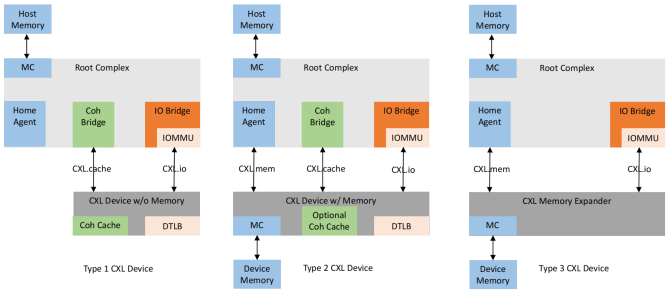



**Figure 1: CXL topology**

### 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 CXL root complex.

Type 1 devices are to use CXL.io or CXL.cache to control the control plane of the PCIe devices, and use CXL.cache to issue the data plane, one unique device is the crypto accelerator that shares some of the cacheline with the program in the CPU. Type 2 devices give all three functionalities, the unique device here is GPU. Type 3 devices provide with CXL.mem and CXL.io which means it's just a memory device.

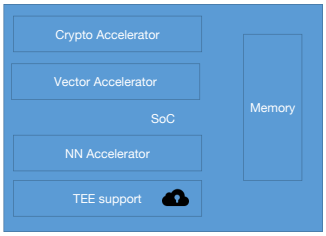



**Figure 2: SoC topology**

## 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 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 0a.

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[6] 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.[11]
(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 exclusiveness-able and will transfer the calculation to kernel's software implementation if we detect the accelerator is in mal state or unplugged.

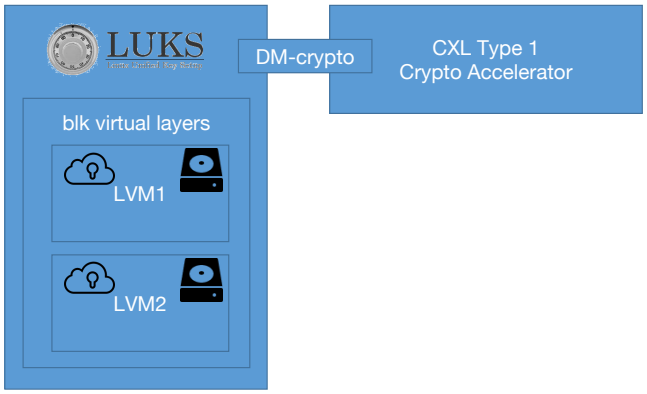



**Figure 3: LUKS topology**

## 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[**?** ] 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[5] 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.

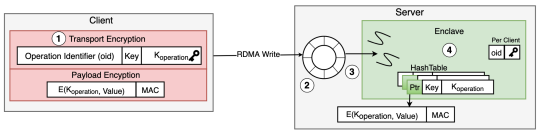



Figure 3: Precursor system architecture. The figure shows the idea of clients precursors and the server-side state

**Figure 4: Precursor Implementations**

## 5.2 QEMU implementation

We first implement the naive CXL devices that run basic type-1 CXL.cache operations. Then we see the current mechanism inside the qemu and the kernel, we try to construct the state that may hurt the IO stack if we offload the LUKS crypto part to the crypto accelerator and do our hardware-software co-design way of mitigating the sensitive state to crash. Our design will be attacking the original un-modified CXL devices and see the possible extra vulnerability that CXL.cache introduce. Then we propose our ATS bit mitigation and the perfect solution so that we can fall back to a slower path once the CXL device is in mal state.

## 5.3 Real Device implementation

We will fully implement an eBPF co-processor near CXL devices. That will help deal with profiling and security. We can filter all the cacheline requests in the CXL bus level and help filter the request to fight against the vulnerability to the kernel.

# 6 PRELIMINARY IDEA FOR VERIFIED DB CXL MEMORY EXPANDER OVER TDX

We think the concerto[1] makes verified steps on a single tree. [7]'s idea can be implemented distributedly using RDMA that the remote devices' enclave can also be leveraged. So we still use merkle hash table, but we offload the verified process onto the device on eBPF processor and enlarge the verifed memory by the local enclave with 1TB TDX per key.

## 6.1 Real Device implementation

We can leverage the aforementioned eBPF co-processor near CXL devices. We can have a naive implementation that put server algorithm in the server and the client algorithm on each eBPF processor.

# 7 RELATED WORK

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

(a) The fuzzing technique [9], recently published on USENIX Sec23, that a user space driver issue can trigger kernel space shuts. And the author designed fuzzing techniques for real-world PCIe device's malicious patterns to better hit the kernel's bugs. They found SoC attached devised with IOMMU has some malicious state.

(b) Other research uses a PCIe firewall protecting every DMA access to the PCIe devices through IOMMU and DMA controller will be filtered by the hook.

**Algorithm 1:** Secure `put()` client-side

**State:**
Operation identifier: `oid`;
Encryption key: $K_{operation}$;
Session key: $K_{session}$;
Encrypted value: *v;
ptr: pointer to the untrusted item;

**Function** `Send_request(`$v$, $key$`)`**:**
  $K_{operation}$ ← KeyGen(); //generate key
  *v← E($K_{operation}$, v);
  mac ← MAC($K_{operation}$,*v ) ;
  oid=oid +1;
  control_data ← ($K_{operation}$, key, oid);
  buf ← $auth - encrypt(K_{session}, control\_data)$;
  $request$ ← (buf, mac, *v) ;
  send $request$ to the server ;
  **return**;

**Figure 5: Precursor client algorithm**

**Algorithm 2:** Secure `put()` server-side

**upon receiving** $request$ **do**
  (buf, payload_data) ← $request$;
  control_data ← $auth - encrypt(K_{session}$,buf);
  ($K_{operation}$, key, oid) ← control_data;
  **if** $oid = C_i.oid$ **then**
    $C_i.oid$ ← $oid$ +1;
    ptr ← $store\_to\_untrusted$ (payload_data);
    $putIntoHashtable(key, K_{operation}, ptr)$;
  **else**
    // Error handling
**end event**

**Figure 6: Precursor server algorithm**

(c) The side-channel attacks have a great vulnerability to the current cloud FPGAs[4], it will be even worse if we extend everything with a cache coherence manner.[11].

(d) The ioctl[4] inside the Linux kernel for PCIe attached devices like FPGA etc. has a timing difference for side-channel attacks evaluation.

(1) Concerto[1, 2, 7] introduced a verified key value that in every working thread will send verified memory access from the enclave to the untrusted host which stores a Merkle hash tree. They also apply deferred verification that after a million instructions were executed on the tree operation, they verified the integrity afterward.

# 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] A. Dubey, R. Cammarota, A. Varna, R. Kumar, and A. Aysu. Hardware-software co-design for side-channel protected neural network inference. *Cryptology ePrint Archive*, 2023.

[4] 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.

[5] R. Hat. Configuring luks: Linux unified key setup, 2023.

[6] Intel. Cxl specification, 2022.

[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] M. Tan, J. Wan, Z. Zhou, and Z. Li. Invisible probe: Timing attacks with pcie congestion side-channel. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 322–338. IEEE, 2021.

[11] 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.