Hardware-Based Methods for Memory Acquisition

Analysis and Improvements

R. van Leenen

Hardware-Based Methods for Memory Acquisition

Analysis and Improvements

by

R. van Leenen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 23rd, 2021 at 10:00 AM.

Student number:	4453530	
Project duration:	September 1, 2020 – March 1, 2021	
Thesis committee:	Dr. Ir. M. Taouil,	TU Delft, supervisor
	Prof. Dr. Ir. S. Hamdioui,	TU Delft
	Dr. Ir. T.G.R.M. van Leuken,	TU Delft
	N. van Heijningen,	Netherlands Forensic Insitute

This thesis is confidential and cannot be made public until August 2023

Preface

This thesis concludes my MSc Computer Engineering at the TU Delft. I would like to thank the people who helped me through the last year.

First, I would like to thank my supervisors from the TU Delft, Dr. Ir. M. Taouil and Ir. M. Beusekom for having seemingly endless patience and for always providing feedback and different perspectives on the project.

Second, I would like to thank N. van Heijningen and J. Rongen from the Netherlands Forensic Institute. Their flexibility and resources provided a way for me to work on the thesis, even during a pandemic. Their expertise and professionalism showed me I have still a lot to learn and made me excited for the future.

Last, but not least, I would like to thank my friends who kept me entertained, even while only online, and my family for their support, patience, and enduring the sound of a serverblade in the room next to them.

I hope you enjoy reading this thesis.

Thank you

R. van Leenen
Delft, July 2021

Contents

Glossary

ACPI Advanced Configuration and Power Interface
General Interface for discovering and interacting with hardware devices on a computer system.

ATS Address Translation Services
PCIe feature used to cache memory translations on the device itself. This allows the device to use translated addresses without consulting an MMU.

CRC Cyclic Redundancy Check
Basic method of reducing data to a small checksum, used to verify data integrity after transmission.

DIMM Dual In-line Memory Module
Common form factor for a memory module.

DMA Direct Memory Access
PCIe feature allowing peripherals to directly and independently access memory without processor intervention.

DMI Direct Media Interface
Interface between the CPU and the PCH.

HDD Hard Disk Drive
Permanent storage device based on a spinning disk.

Hot-Plug
The act of inserting a (new) device into a computer system while the system is still powered-on (hot).

I/O Input/Output
Indicates any sort of communication going into or out of the computer system.

IntelME Intel Management Engine
Security Subsystem within the PCH.

KASLR Kernel Address Space Layout Randomization
Technique for randomizing the location of the kernel in memory.

LFSR Linear-Feedback Shift Register
Linear sequence generator, commonly used to generate pseudo-random numbers.

LPC Low Pin Count
Low bandwidth I/O bus found on computer systems.

MMU Memory Management Unit
Memory controller residing in the CPU. It provides bus arbitration, cache control, and memory protection through virtualization.

MOS Metal-Oxide Semiconductor
Production process for making a.o. transistors.

MPU Memory Protection Unit
Smaller version of the MMU, provides only basic memory protection.

MTRR Memory Type Range Registers
CPU feature that allows software to control how accesses to certain memory ranges are cached.

NIC Network Interface Controller
A hardware component used to connect a computer system to a network.

OS Operating System
General software allowing users to interact with the underlying hardware.

PAT Page Attribute Table
CPU feature that allows software to control how accesses to certain pages are cached.

PCH Platform Controller Hub
Intel-Specific I/O hub, similar to a southbridge.

PCI Peripheral Component Interconnect
A High-Speed bus used for connecting peripherals to the computer system. Followed up by the improved PCI Express (PCIe).

RAID Redundant Array of Independent Disks
Feature of Storage Controller that utilizes multiple independent disks to create redundancy in storage in order to prevent and recover from faults in permanent storage devices.

RAM Dynamic Random Access Memory
A specific type of RAM technology which utilizes a capacitance on the semiconductor fabric to store the data.

RAM Random Access Memory
The memory a computer system uses to store data temporarily.

SATA Serial Advanced Technology Attachment
Communication bus typically used for storage devices.

SPI Serial Peripheral Interface
A serial I/O interface found on computer systems.

SSD Solid State Drive
Permanent storage device typically based on flash storage.

TLB Translation Lookaside Buffer
A cache which caches Virtual to Physical Address translations.

TLP Transaction Layer Packet
A PCIe packet used on the transaction layer. Used for a.o. sending DMA requests.

VM Virtual Machine
Virtual emulation of a computer system within another computer system.

Abstract

Dutch server hosters contribute significantly to (worldwide) cybercrime. For example, a report by the Internet Watch Foundation published in 2019 indicated that 71% of worldwide child sexual abuse content is hosted in the Netherlands. Due to the good IT-infrastructure and favourable investment climate, the Netherlands attracts a lot of server hosters, including ones with ill intentions. Some server hosters facilitate cybercrime either intentionally (so called “bulletproof hosters”) or unintentionally (“bad hosters”). When dealing with uncooperative hosters during forensic investigations, it may sometimes be necessary to collect data or information on the servers without help from the owner of the server.

Data within the RAM might prove insightful in, for example, determining active processes or reveal cryptographically interesting information like encryption keys. The thesis explains key concepts within memory organization and the PCIe standard. Afterwards, it discusses several techniques for RAM acquisition and categorizes and evaluates them using a model-based approach. The thesis then dives deeper into DMA-based memory acquisition using PCIe and proposes several improvements to current DMA attacks in order to create a better memory acquisition technique. A novel memory acquisition technique is created by hot-plugging a PCIe device and skipping over the regular enumeration procedure. This technique allows the memory acquisition to be executed without a reboot and provides a stealth approach to accessing the memory.

1 Introduction and General Overview

This chapter provides an introduction to this thesis. It first describes the motivation behind the thesis in Section 1.1, followed by the context of digital forensics to which this project developed in Section 1.2. After a state-of-the-art analysis in Section 1.3, it concludes with a description of the contribution of the thesis and the general structure for the rest of the thesis in Sections 1.4 and 1.5 respectively.

1.1. Motivation

On June 29th 2020, the Dutch Minister of Justice and Security released a progress report [64] about cybercrime and the ministry's resulting plan of action. The report discusses Dutch server hosters and their contribution to (worldwide) cybercrime. Due to the good IT-infrastructure and favourable investment climate, the Netherlands attracts a lot of server hosters, including ones with ill intentions. These hosters facilitate cybercrime either intentionally (so called "bulletproof hosters") or unintentionally ("bad hosters"). When dealing with uncooperative hosters during forensic investigations, it may sometimes be necessary to collect data or information on the servers without help from the owner of the server.

The first thing to come to mind when talking about data storage is the hard drive, but hard drives are not the only relevant type of data storage on the server. Data within the RAM might prove insightful in, for example, determining currently active processes or reveal cryptographically interesting information like encryption keys. This thesis will discuss techniques for acquiring the data within the RAM.

Manufacturers are continuously improving security measures in their servers and techniques for forensic acquisition of memory data that worked previously may not be effective today. Modern solutions may have shortcomings that need to be fixed before they can be utilized effectively, or an entirely new solution for RAM-data acquisition may need to be developed. Either way, an effective solution for acquiring the data will lead to more effective forensic investigations.

1.2. Digital Forensics

In order to better understand the context of this thesis, it is important to understand the forensic background of this project.

Forensic Process

Digital devices are becoming more commonplace in everyday lives by the year. This also means that digital evidence found on these devices will take a more common role in criminal court cases. Since digital devices can play a vital role in criminal investigations and provide many opportunities for collection of evidence, it is of utmost importance that that evidence is applied correctly within the courtroom. A properly validated scientific method for the process of collecting evidence is fundamental to a sound forensic process. There is no standard forensic process for collecting digital evidence, although many have tried to create one over the years [26] [10]. These methods mostly share three common stages within the forensic process:

- **Seizure** The collection of the medium on which the evidence may be located.
- **Acquisition** The acquisition of the raw data of the evidence.
- **Analysis** The analysis of the acquired data.

In this thesis, only the second step of the process is considered.

Forensic Soundness For the results of this thesis to be used in forensic research, it is important that the acquisition method is forensically sound. Vömel and Freiling [68] define three criteria for forensically sound memory acquisition. These criteria are as follows:

- **Correctness** The snapshot data should reflect the original data stored in the memory. Correctness of the memory acquisition method can be reduced by, for example, the method leaving a footprint in the snapshot, or bitwise errors caused by the acquisition method.
- **Atomicity** The snapshot should be free of concurrent memory activity. Atomicity of the memory acquisition method can be reduced by, for example, the method not preventing writes to memory.
- **Integrity** The snapshot data reflects the memory data at that timestamp correctly. Integrity of the memory acquisition method can also be reduced by, for example, the method not preventing writes to memory.

These are idealized and abstract criteria, but they are nonetheless useful when determining relevant metrics to evaluate the differences between acquisition methods.

1.3. State-of-the-Art Memory Acquisition Techniques

RAM started to become more forensically interesting at the beginning of the 2000's. Volatile DRAM (more in Section 2.3.2) was and still is the default temporary memory storage and with the surge of consumer computer systems, two main use cases for RAM acquisition started to appear [18]. The first use case was for investigating and countering a new type of computer virus. Worms like Code-Red [47] and SQL Slammer [61] were worms that resided solely within the RAM. The appearance of these worms coincided with rising interest with regards to the second use case. Digital forensic investigators at the time focussed mainly on permanent storage devices. Studies showed that the RAM might also contain interesting information like passwords and the state of the system at seizure [19]. Already at this time, the main acquisition technique was DMA (Direct Memory Access, more in Section 2.5.2) over PCIe (more in Section 3) [18].

Soon after, software tools to acquire memory data on several different OSs were developed [67]. These tools all suffered from the same drawbacks. They were run on the target system itself, which spoiled a clean RAM acquisition with data from the tool itself. Apart from that, anti-forensic measures were slowly becoming more commonplace [57]. These countermeasures have, in fact, still not fully countered memory access using software tools. Even to this day, software tools to acquire a memory image on the target PC are still fully working and available (more in Section 4.2).

Hardware based (or assisted) methods were considered superior due to DMA not leaving a footprint in memory and the lack of need for user access to the system [54]. DMA using hardware interfaces was already easy with PCI, but the introduction of IEEE 1394, also known as FireWire, also allowed for easy access to a DMA-capable port [2]. Eventually PCI was superseded by PCI Express (more in Section 3), which continued to support DMA. FireWire did not gain much traction outside of Macintosh systems, and was eventually phased out completely. PCIe, however, is present on most, if not all, modern computer systems. Its high-speed and access to the memory has led to several memory acquisition toolkits being developed for PCIe (more in Section 4.3). The emergence of Thunderbolt, which supports PCIe and therefore DMA, may open new avenues for PCIe-based vulnerabilities or DMA based memory access [59] [55] [23] [40]. Over the years, several tools have been developed to perform ”DMA Attacks” over FireWire (Inception [46]), PCIe (PCILeech [30]), or Thunderbolt (Thunder-Spy [55], or ThunderClap [59]).

A second hardware-based method emerged around 2008, which took a more direct approach to acquisition of volatile memory. Cold Boot attacks bring the volatile memory to a lower temperature [38] [37]. This lower temperature allows the memory to retain the data for longer periods of time. This allows for a transition period in which the system can be rebooted to a new OS that can extract the memory data. The memory device can also be transplanted into another computer system for further acquisition. A third acquisition technique that works on the same level would connect directly to the DRAM using the memory bus [9].

The main countermeasure against DMA attacks is the IOMMU (more in Section 2.4.2). The IOMMU promised to prevent DMA attacks, which is a promise on which it has delivered for the most part [59]. But the sentiment in 2007, shortly before the release of Intel’s IOMMU, which was that there would be a ”goodbye to hardware based memory acquisition”, is not entirely true [5] [54]. This is mostly due to the poor adoption of the IOMMU by OS vendors. Nowadays, most hardware manufacturers have some sort of IOMMU implemented in either the CPU or the chipset [69].

Both hardware and software based acquisition methods can be countered using RAM encryption [48]. The key to a succesful RAM encryption method is the protection of the encryption key. Should the key be compromised, the encryption becomes useless [15]. A second encryption mechanism can be found directly on the memory bus. Intel’s memory scrambler succesfully defeats most hardware attacks that target the memory bus by scrambling the data over the bus using an LFSR [11].

1.4. Contribution

The primary goal of this thesis is to find a technique for RAM acquisition on server systems. The main contributions of this thesis are:

- **A classification and overview of techniques for RAM acquisition**
  A classification of modern popular acquisition techniques based on the interface at which the memory data can be acquired. An analysis of the main strengths and drawbacks of each technique results in an good overview of the current acquisition techniques.
- **A model-based approach to determining which techniques are (or can be) effective**
  A model, adapted for memory acquisition techniques, designed to determine the most promising acquisition technique, .
- **An overview of the main limitations and drawbacks of memory acquisition using DMA attacks with an inserted PCIe device**
  An overview of the limitations and drawbacks of this technique provides a clear list of focus points in order to make memory acquisition using this technique more attractive for use in digital forensics.
- **Several proposals for improvement of memory acquisition using DMA attacks with an inserted PCIe device**
  Several, including unique and novel, proposals for improving the acquisition technique, solving some of the limitations and drawbacks from which it currently suffers.
- **A demonstration of a successful memory acquisition using DMA attacks with an inserted PCIe device**
  A full demonstration of a DMA attack with an inserted PCIe device, including scenarios in which current DMA attacks would fail. The improved technique results in a wider range of scenarios in which the technique is effective and increased target system stability during application.

1.5. Structure Overview

This thesis is divided into 7 chapters.

- **Chapter 1: Introduction**
  The motivation behind the thesis is described, followed by the context of digital forensics. After a state-of-the-art analysis, it concludes with a description of the contribution of the thesis and the general structure for the remainder of the thesis.

- **Chapter 2: Memory Organization**
  Here, an overview of modern memory organization is provided by discussing the relevant concepts with regards to the three main building blocks of modern day storage: the cache, the main memory and permanent storage. It also discusses memory controllers and their impact on computer system memory, as well as the interfaces which can connect to the DRAM.

- **Chapter 3: Peripheral Component Interconnect Express**
  This chapter discusses the basics of PCI Express, and Direct Memory Access (DMA) using PCIe.

- **Chapter 4: Memory Acquisition Techniques**
  This chapter discusses the different existing techniques or concepts of techniques for acquiring memory. It classifies them into three categories based on the interface where the acquisition is executed. Next, a model is used to determine to most effective method of memory acquisition. This chapter discusses the reasoning behind the model, applies it to available hardware methods, and discusses the result.

- **Chapter 5: DMA-based Memory Acquisition Using PCIe**
  In this chapter, it is described what a DMA attack exactly consists of. It first discusses existing DMA attacks and the drawbacks of the attacks. Based on a vulnerability analysis of PCIe and DMA, some improvements to the memory acquisition method are proposed.

- **Chapter 6: Validation and Results Analysis**
  This chapter discusses the platform used to experiment on the PCIe bus, as well as perform the memory acquisition. It describes the device used to execute the acquisition and the changes that were made to the device, based on the proposed improvements in Chapter 5.

- **Chapter 7: Conclusion**
  Finally, the conclusion provides a summary of the thesis and recommendations for future work.

2
Memory Organization

This chapter starts with an overiew of modern computer organization in Section 2.1, based upon which it explains an overview of modern memory organization in Section 2.2, discussing the relevant concepts with regards to the three main building blocks of modern day storage in Section 2.3: the cache, the main memory and permanent storage. It also discusses memory controllers and their impact on computer system memory in Section 2.4, as well as the interfaces which can connect to the DRAM in Section 2.5.

2.1. Modern Computer Organization

In order to understand more about memory and its place within a computer system, it is important to discuss general computer organization first.

As seen in Figure 2.1, the CPU and the chipset form the centerpiece of any computer system. The CPU provides all necessary computing power, while the chipset facilitates the interaction for peripherals with that computing power. The main memory is connected to the CPU using the Memory Management Unit (MMU) and most peripherals to the southbridge, which has a direct connection to the CPU using an internal bus.

Current implementations of the southbridge function mostly as a I/O hub, providing connections for common busses such as SPI, LPC, SATA, or PCIe, but may also contain other devices like IntelME which provide extra functionality. The northbridge is in charge of controlling the memory and providing connections for high-speed peripherals (graphics cards over PCIe). Previously, the northbridge and southbridge were two separate chips on the motherboard, but over time the northbridge got absorbed by the CPU including all corresponding functionality starting with the memory controller in Intel's Nehalem and AMD's AMD64 processors and eventually completely in Intel's Sandy Bridge and AMD's APU's [39]. In more recent processor architectures by AMD, the southbridge is slowly facing the same fate as well [36].

As can be seen in Figure 2.1, the PCIe bus is an interesting case: it is both used for high-speed graphics (which prefers a direct connection to the CPU, possibly connected using an IOMMU) as well as regular peripherals (which can be connected to the southbridge). Therefore the PCIe bus can be connected to the southbridge, the CPU or both at the same time, depending on the platform.

2.2. Modern Memory Organization

When talking about memory, there is a traditional distinction between primary (main) and secondary (auxiliary) storage [1]. Secondary storage is not directly accessible by the CPU. It is slow, but contains large, permanent storage like SSDs or HDDs. Primary storage is directly accessible by the CPU. It is fast and small. It is usually also volatile: once the system loses power, the data on these storage

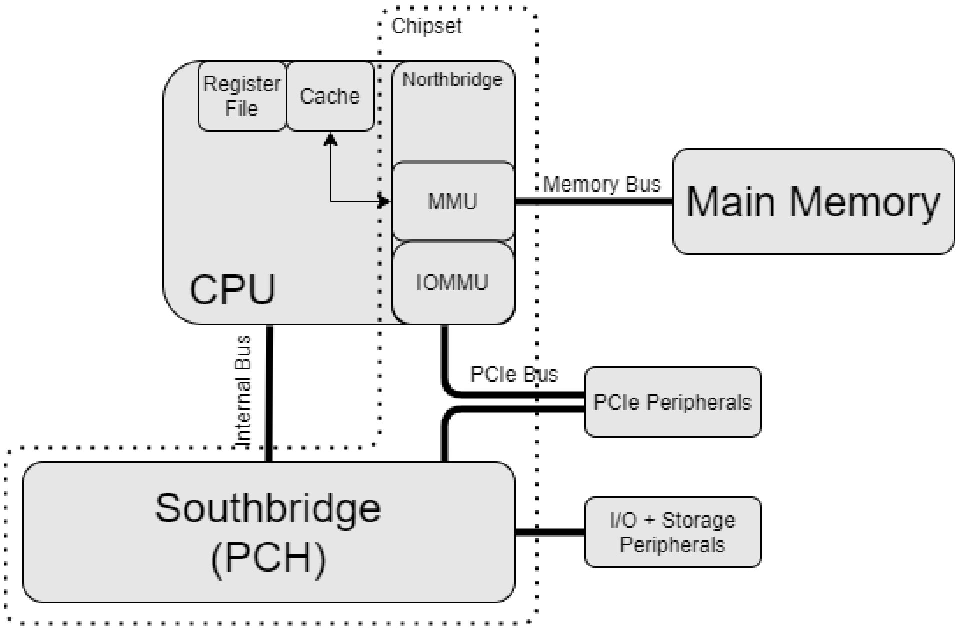


Figure 2.1: Modern General PC Architecture

devices is lost. The exception to this is the Boot ROM, which contains the starting instructions that a system uses to boot from. Figure 2.1 displays the location of the different types of storage discussed below. The secondary storage is part of the storage peripherals connected to the southbridge. The primary storage is divided into the following three parts:

- **Register File** The register file is used to store data (typically in the order of bytes) inside your CPU, used for immediate execution. The content of the register file is determined by the program that is currently running.
- **Processor Cache** The processor cache is used to store kilobytes of data within your CPU. The cache is managed by the MMU, which determines which parts of your RAM are copied to the cache, for faster access by the CPU.
- **DRAM** The DRAM is used to store gigabytes of data outside of the CPU, and is managed by the MMU as well. Not only the CPU, but also peripheral devices can access the DRAM.
- **Boot ROM** The Boot ROM is a type of non-volatile primary storage. It stores kilobytes of data outside the CPU. It contains the starting instructions that a system uses to boot from.

An overview of the different types of storage can be seen in Figure 2.2. As the size of the storage decreases, the latency to access the data decreases, the power required to store the data increases, as well as the area per storage unit.

The MMU (discussed in Section 2.4) has (full) control over three of the aforementioned storage elements: The cache (Section 2.3.1), the RAM (Section 2.3.2), and permanent storage devices (Section 2.3.3). The register file is too small to hold any forensically interesting data and will not be elaborated on. The Boot ROM only contains instructions and is therefore also excluded.

2.3. Storage Types

The storage types presented in the previous Section (2.2) are described below.

2.3.1. Cache

The cache resides within the CPU and is implemented in order to have faster access to primary storage. The cache accomplishes this speed-up based on two principles:

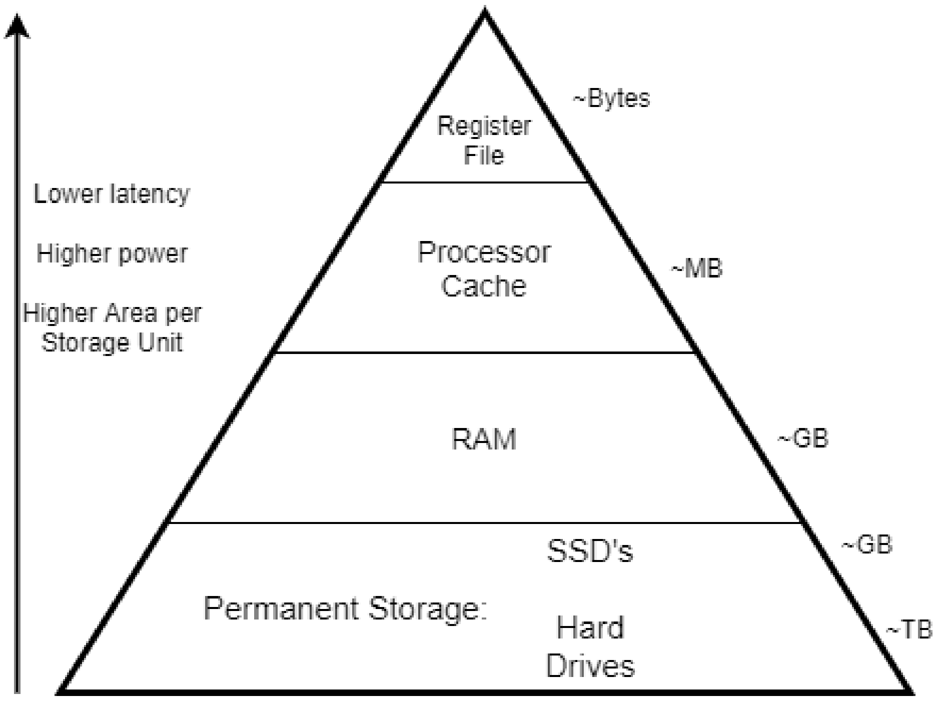


Figure 2.2: Layered Storage Model

Spatial Locality: If some data has been used, it is likely that the data next to it will be used as well. By bringing this data to the cache, one can use this data immediately, without having to wait for it as long as the previous request.

Temporal Locality: If some data has been used, it is likely that the data will be used again at a later date. By keeping this data stored in the cache, one can use this data later immediately, without having to wait for it as long as the previous request.

Based on the address of the requested data, the correct piece of data can be collected from the cache. If it is missing, the data may reside in the DRAM or in permanent storage. Once the data is collected, it will be placed inside the cache for (possible re-)use.

Apart from the size of the cache, the most important attribute is the associativity of the cache. In a direct mapped cache each memory block is only mapped to one line in the cache. In a set associative cache, one cache line can contain multiple different memory blocks. If a memory block is to be placed on a cache line which already contains the maximum amount of blocks, one will have to be evicted based on a replacement policy. A cache which only contains one line is called a fully associative cache. Although a direct mapped cache is faster, a set associative cache can decrease cache misses.

The data in the cache is not always perfectly coherent with the data found in the DRAM. When the CPU changes data in the cache, this data would need to be written back to the DRAM as well. The reverse is also true, if data is changed in the DRAM, the cache would need to be changed as well. Features like MTRR (Memory Type Range Register) [33] or PAT (Page Attribute Table) [44] can also influence the way data is written back/through to the cache or DRAM. These processes can control memory accesses more directly by, for example, forcing a write-through to the DRAM. This cache coherency problem makes acquiring ”memory” for forensic purposes difficult. Determining if the memory found in the physical DRAM represents the current state of the system is a hard task.

A cache can consist of multiple levels, usually denoted as L1, L2, etc. Each level of cache increases in size, but comes with a lower speed with which the cache can be accessed. This level-based structure of cache relates the frequency of access of the data to the penalty for looking up that data. Lower levels of cache are implemented as Static RAM (SRAM), while higher levels may sometimes be implemented using DRAM as well. Different types of DRAM are discussed in Section 2.3.2.

2.3.2. Main Memory

The main memory of the computer system is volatile memory stored outside of the CPU. The most common form-factor is a DIMM(dual in-line memory module) or RAM-stick of DRAM like the one seen in Figure 2.3.

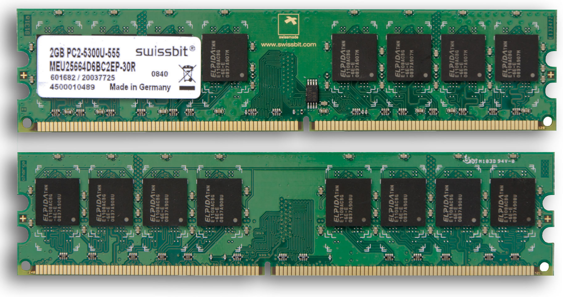


Figure 2.3: Front and backside of a 2 GB DDR3 DIMM [8]

The history of RAM on Metal-Oxide Semiconductors (MOS) started as static RAM, in which each bit is stored using latching circuits. Since this was very area expensive, dynamic RAM was developed which utilized a capacitance on the semiconductor fabric to store the bit [6]. Since the charge on this capacitance slowly leaks away (partially due to the transistor leakage current), the charge needs to be refreshed periodically to maintain a positive value.

As can be seen in Figure 2.3, DRAM does not consist of one big chip, but several smaller ones, each controlled separately. Memory on a memory stick is often interleaved. In order for memory requests with a similar spatial locality to be completed simultaneously, memory within a similar address range is not placed on the same chip, but distributed among different memory banks on the memory stick, as seen in Figure 2.4. This Figure shows data ordered by memory address on the left, and the location the data on four different chips on the DRAM on the right. This makes memory lookups within a similar memory range more likely to be able to be looked up in parallel.

Some system peripherals also contain addressable memory regions. To make these regions accessible by the CPU and other peripherals, they take up part of the memory space. This concept is called Memory mapped I/O (MMIO). Access to that memory is then relayed to the corresponding peripheral by the MMU.

Some devices on the motherboard do not have memory of their own, but would still like to use memory. These devices may claim a memory region for themselves and do not allow access to that region by other devices. Examples are the BIOS or IntelME (Intel Management Engine, used for system security). These memory regions can be found at the Top of Low Usable DRAM (TOLUD). This is at 4GB and the memory regions downward from 4GB will be stolen. Usage of these memory regions by other programs is (completely) restricted.

2.3.3. Permanent Storage

Permanent storage is non-volatile storage generally located outside the CPU. Examples are HDDs or SSDs, but also USB storage devices or cassette tapes. The storage content can be determined by a.o. the user, the OS, the MMU, and a storage controller. Accesses to permanent storage devices are

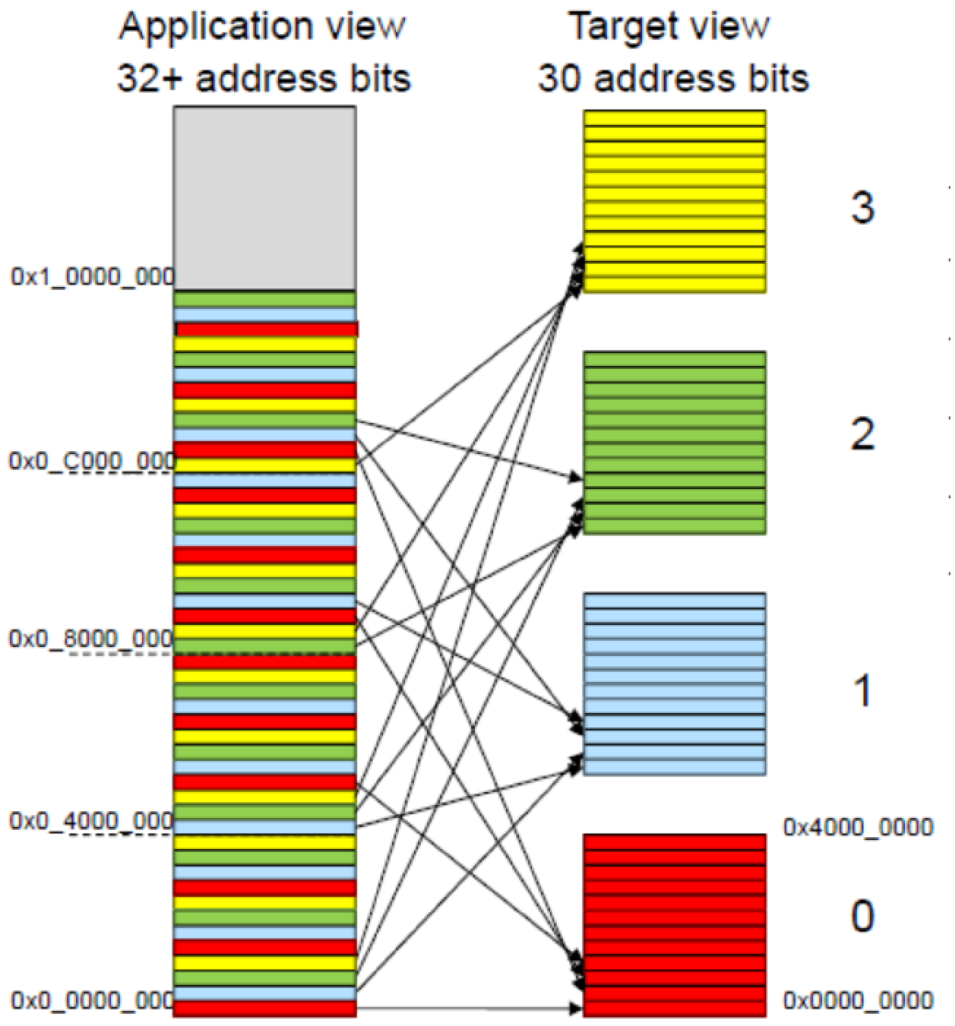


Figure 2.4: Interleaving of data onto several chips by Grammatikakis et al. [34]

extremely slow compared to the primary storage.

A common type of permanent storage is a Hard Drive Disk (HDD). The data is stored on a magnetic rotating disk. More recent and faster variants are Solid State Drives (SSD), which store the data on flash memory. These storage peripherals connect to the system using the southbridge or a similar IO-hub on the PCIe bus. SATA is a commonly used bus to connect the peripheral to the system. Since SATA is bottlenecked by its bandwidth due to a low number of pins, other form factors like mSATA or M.2 (which work over PCIe) have gained more traction in recent years.

Both the user and the OS can control files on a permanent storage device. However, permanent storage devices can also contain part of the main memory. The MMU (further described in Section 2.4.1) might store data in permanent storage in a paging file. The MMU handles paging as part of virtualization (more in Section 2.4). Another device which can control the data on permanent storage is a storage controller. A storage controller can implement features such as RAID (Redundant Array of Independent Disks), which utilizes multiple independent disks to create redundancy in storage in order to prevent and recover from faults in permanent storage devices.

2.4. Memory Controllers

Memory controllers provide the interaction between the memory and the system. Over time the north-bridge (which handled memory interactions) was absorbed by the CPU and its functionality distributed over several parts within the CPU. The term ”memory controller” now not only includes the physical interaction with the DDR bus, but can also include functionality like virtualization. As long as it provides some sort of functionality with regards to the memory, it is part of the ”memory controller”. The most common form of memory controller is the memory management unit (MMU). Its smaller sibling, the IOMMU, provides similar functionality but for IO peripherals.

2.4.1. Memory Management Unit

The main memory controller in computer systems is the MMU (memory management unit). It resides in the CPU and provides memory protection, bus arbitration, virtualization, and cache control. The functionality of the MMU is implemented in the integrated memory controller devices on an internal bus within the CPU. An example can be seen in Figure 2.5, which shows the Integrated Memory Controllers on an internal bus in an Intel Xeon E5-1600/2400/2600/4600. On the same bus is also the Intel QPI controller, which controls the internal bus from the CPU to the southbridge. Some processors feature a smaller version of an MMU, then called a Memory Protection Unit (MPU).

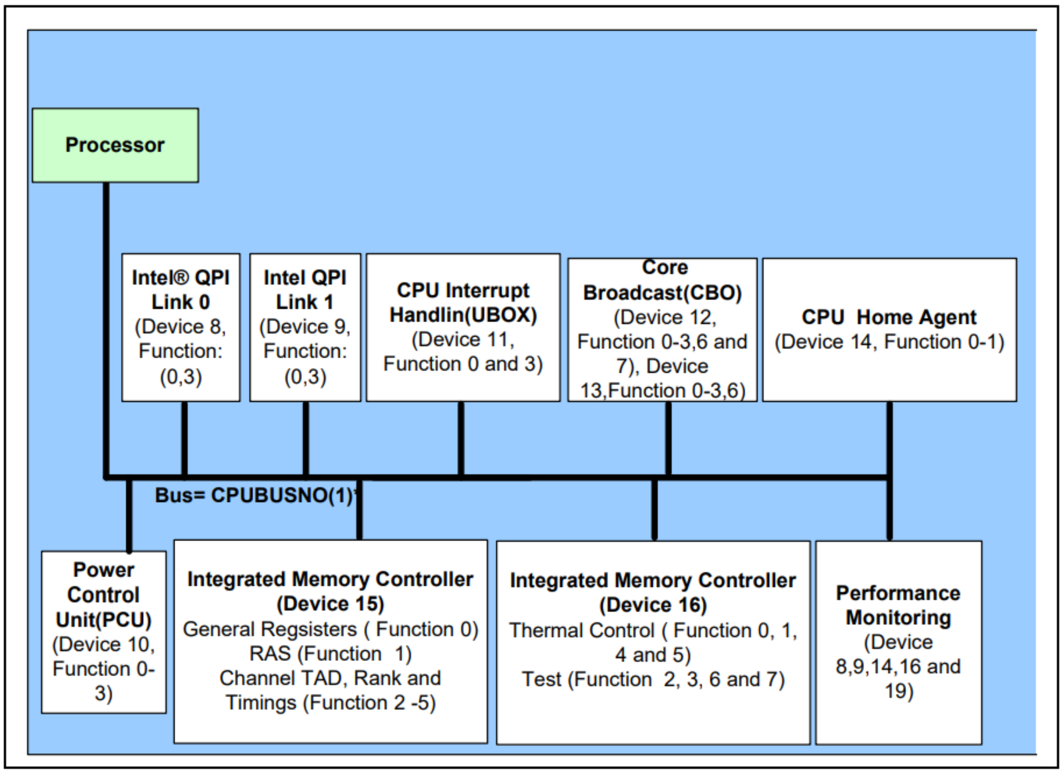

Processor
Intel® QPI Link 0 (Device 8, Function: (0,3)
Intel QPI Link 1 (Device 9, Function: (0,3)
CPU Interrupt Handlin(UBOX) (Device 11, Function 0 and 3)
Core Broadcast(CBO) (Device 12, Function 0-3,6 and 7), Device 13,Function 0-3,6)
CPU Home Agent (Device 14, Function 0-1)
Bus= CPUBUSNO(1)
Power Control Unit(PCU) (Device 10, Function 0-3)
Integrated Memory Controller (Device 15) General Regsisters (Function 0) RAS (Function 1) Channel TAD, Rank and Timings (Function 2 -5)
Integrated Memory Controller (Device 16) Thermal Control (Function 0, 1, 4 and 5) Test (Function 2, 3, 6 and 7)
Performance Monitoring (Device 8,9,14,16 and 19)

Figure 2.5: Internal bus containing the integrated memory controller device in the Intel Xeon E5-1600/2400/2600/4600 [22]

Memory virtualization is the process of assigning virtual memory to a separate process running on the PC. By utilizing virtual memory, each process gets assigned a virtual address range in which it can store data. It works by providing a larger memory space than is actually available, which results in a more efficient use of the available memory space. If the main memory is full, it pushes data to a paging file in the secondary storage. A paging table is used to translate virtual addresses into physical addresses and can maintain where data is located. This translation incurs a performance penalty, which has been slightly mitigated using a Translation Lookaside Buffer (TLB). The TLB is a cache that caches frequently used translations from virtual to physical addresses.

The MMU and MPU provide memory protection by linking the requested address to the process that requests it. The MMU will not translate the address (and therefore limit access to the data) if that process should not have access to that data. Memory protection provides mostly protection against mistakes made by a process. It does not provide actual full protection against reading unintended memory ranges.

The MMU also provides bus arbitration for the memory bus. If multiple memory access requests are made, the MMU decides the order in which the requests are handled. An example of a bus arbitration scheme is least recently serviced (LRS), which is implemented in the MicroChip PIC32MZ [42].

2.4.2. IOMMU

The IOMMU provides peripherals with Virtual Address space to access, similar to how a MMU provides processes with a Virtual Address space. DMA accesses (discussed in Section 2.5.2) to that memory region is then limited based on the device's PCIe ID (discussed in Section 3.3). The IOMMU is not a recent development, but it is still not always supported or utilized properly [53]. Vulnerabilities with regards to the IOMMU can be found in Section 5.3.

Implementations and Support
The three most important CPU vendors (Intel, AMD and ARM) all have an implementation for an IOMMU:

- **Intel** - Virtualization for Directed IO / VT-d
- **AMD** - AMD-Virtualization / AMD-V / AMD-Vi
- **ARM** - System MMU / SMMU

Most OSs already provide some sort of IOMMU support, but this may not be turned on by default. BIOS or UEFI should also support turning on Virtualization, which is very common. An overview of OS support can be seen in Table 2.1.

OS:	Windows	Linux	MacOS
Supports Intel VT-d	yes	yes	yes
Supports AMD-v	no	yes	?
Supports ARM's SMMU	no	yes	?
Default On	no	no	yes

Table 2.1: IOMMU support per OS

Virtualization for Directed IO / VT-d This thesis limits itself to describing Linux's implementation of Intel's VT-d, since this combination fits best into the scenario described in Section 4.5.

VT-d is a hardware unit found in Intel processors since 2008 [5]. The main concept of its implementation has not changed drastically over the years. VT-d is found on the internal part of the DMI bus, as seen in Figure 2.6.

The unit consists of multiple DMA remapping hardware devices or DRHDs. Each device provides page-based translation capabilities to translate IO virtual addresses to physical addresses based on the PCIe ID of the requesting device and the requested virtual address. The DRHDs are configured by a concatenation of 6 page tables commonly known as DMA Remapping Tables (DMAR tables) in order to achieve a translation. Since this incurs a lot of overhead, the translations may be cached in the IO Translation Lookaside Buffer (IOTLB), similar to a TLB in a regular MMU.

Devices are grouped by domain: a memory area which is completely separate from other domains. Devices in one domain cannot access the other domain, which is restricted by the PCIe ID. Windows

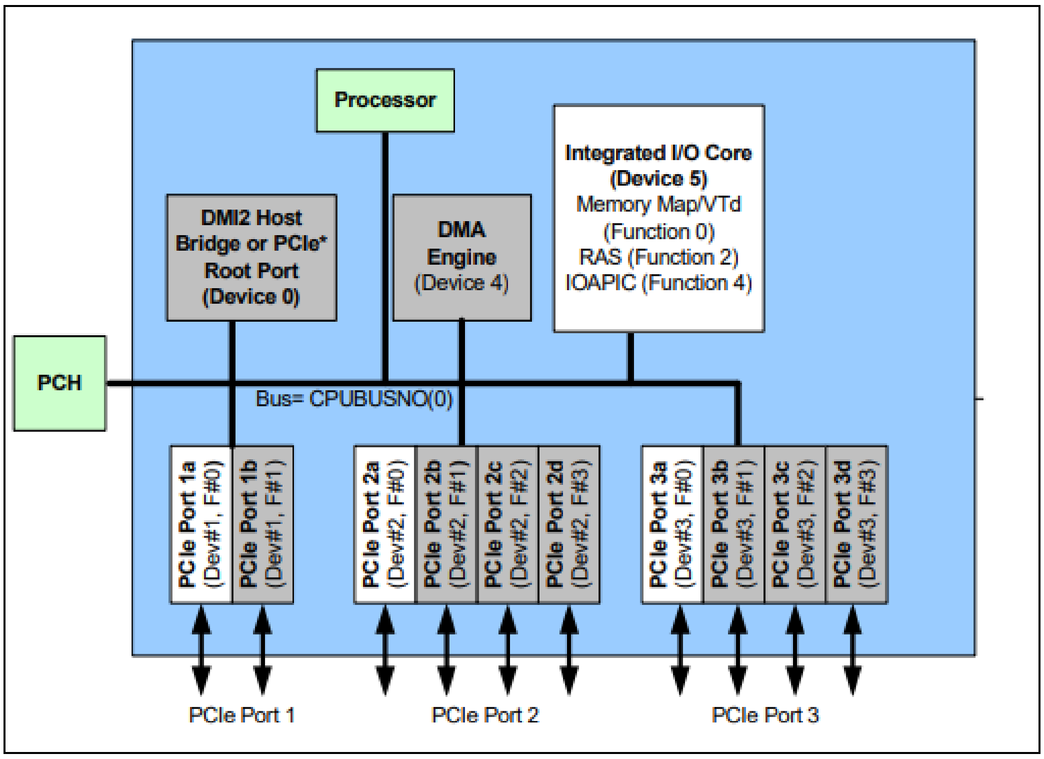

Processor
Integrated I/O Core (Device 5)
Memory Map/VTd (Function 0)
RAS (Function 2)
IOAPIC (Function 4)
DMI2 Host Bridge or PCIe* Root Port (Device 0)
DMA Engine (Device 4)
PCH
Bus= CPUBUSNO(0)
PCIe Port 1a (Dev#1, F#0)
PCIe Port 1b (Dev#1, F#1)
PCIe Port 2a (Dev#2, F#0)
PCIe Port 2b (Dev#2, F#1)
PCIe Port 2c (Dev#2, F#2)
PCIe Port 2d (Dev#2, F#3)
PCIe Port 3a (Dev#3, F#0)
PCIe Port 3b (Dev#3, F#1)
PCIe Port 3c (Dev#3, F#2)
PCIe Port 3d (Dev#3, F#3)
PCIe Port 1
PCIe Port 2
PCIe Port 3

Figure 2.6: Internal Extension of the DMI bus to different devices in the Intel Xeon E5-1600/2400/2600/4600 [22]

groups all devices in the same domain, while MacOS groups all devices in separate domains. Linux can do either based on the IOMMU setup. Certain peripherals can also be set to pass through the IOMMU altogether. This may be necessary for certain graphics-based peripherals, which are uncommon in servers.

2.5. Memory Interfaces

The following section discusses the interfaces used to interact with the main memory.

2.5.1. Memory Bus

The memory communicates to the CPU using the memory bus. The memory bus became more complex over the years in order to facilitate more bandwidth. This started with the introduction of Synchronized DRAM, which synchronized the clocks of the memory controller and the memory. Although this was slightly slower than using unsynchronized memory transactions, it allowed for pipelined memory requests. This pipelining led to an overall increase in performance. Another commonly used form of DRAM is Double Data Rate (DDR) SDRAM. This protocol utilizes both the upgoing and downgoing clock flank to send data, making it a complex protocol to analyze and/or utilize. In order to reduce latency, the memory bus provides a direct connection to the DRAM chips. The standard for DDR is maintained by JEDEC [43], which is an association with many chip manufacturers as its members.

The controller of the DDR bus is implemented within the Memory Controller. The Intel Memory Scrambler also scrambles the data on the memory bus with a pseudo random LFSR. This leads to more uniform data, which means less power spikes, which in turn means better signal integrity. It also functions as a countermeasure to attacks that target the memory via the memory bus [11].

2.5.2. Direct Memory Access

Before Direct Memory Access (DMA) existed, the CPU was the only authority that could transfer data from the memory to a peripheral device. This placed a huge load on the CPU, since for every bit of data that a peripheral device needed to read or write from memory, it needed to orchestrate the entire transfer. DMA gives peripheral devices the power to directly access the memory, freeing up CPU resources. DMA can theoretically happen over any IO bus. The most common protocol which supports DMA is PCIe. More on PCIe and detailed information about DMA over PCIe can be found in Section 3.5.

The process of DMA is started either by the CPU or the peripheral device itself. The CPU can use a DMA engine, which is implemented within the CPU as seen in Figure 2.6. This DMA engine can control the DMA transfer between a device and the memory or two memories on two separate devices. A DMA engine is used when the device is incapable of performing DMA itself, or when memory between two separate computer systems needs to be shared. A peripheral device can also initiate the DMA process by themselves. Peripheral devices are assigned a flag called ”Bus Master”. If this flag is set, the device is allowed to access the memory and perform DMA. The reverse is not true: a device without the bus master flag enabled is not prohibited from accessing the memory.

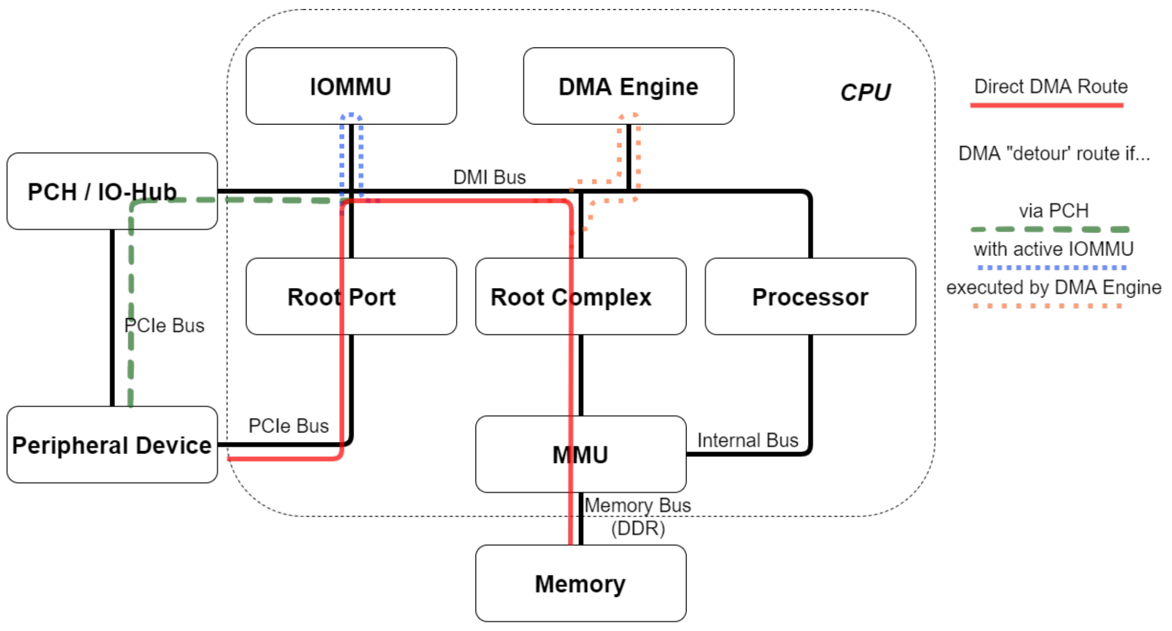


Figure 2.7: DMA flow with multiple possible paths

The flow of DMA can vary, depending on who starts the transaction and on top of which protocol DMA is run. An example of different DMA flows can be seen in Figure 2.7. This figure shows an example of DMA using a peripheral connected over PCIe, which is accessing the main memory of the system it is connected to. If the peripheral is connected to the system through a Root Port of the Root Complex (more on these PCIe specific devices in Section 3.4). The Root Complex is connected to the MMU, which is connected to the memory. The device might also be connected through the PCH, which merely entails a different routing and no functional difference. If the IOMMU is active, the flow changes to include the IOMMU as well which can, for example, translate the address of the memory access. If the device is incapable of performing DMA itself, the DMA engine can control the flow. The DMA engine can be instructed to do so by the processor. Note that in Figure 2.7 there is still a direct connection between the MMU and the processor. Regular memory accesses by the processor happen through this internal bus which can also include e.g. communication to the cache.

The specifics of memory accesses utilizing the functionality within the PCIe protocol which enables DMA can be found in Section 3.5. As DMA is an easy way to access the main memory, the DMA process can be misused my malicious actors in order to gain access to the system. This is called a DMA attack. Examples of different DMA attacks can be found in Section 4.3. A concept of a DMA attack using PCIe can be seen in Chapter 5.

3 Peripheral Component Interconnect Express

This chapter discusses the basics of PCI express and Direct Memory Access (DMA) using PCIe. A full overview of the entire setup can be seen in Figure 3.3 at the end of this chapter. This Figure is referenced throughout this chapter. Important busses or connections are indicated with lines, important processes over those lines are indicated with arrows. Section 3.1 starts with an introduction to PCI and PCIe, after which the different protocol layers are explained in Section 3.2. Next the details of PCIe peripherals and the root complex are explained in Sections 3.3 and 3.4 respectively. Finally, the basis of DMA using PCIe is discussed in Section 3.5.

3.1. Introduction to PCI and PCIe

Peripheral Component Interconnect, or PCI, is a communication protocol developed for connecting peripheral devices on a computer system. Its successor, PCI Express (PCIe), is currently very common in any type of modular system. The standard for PCI and PCIe is maintained by PCI-SIG [50].

PCIe is a high-speed serial connection bus, over which a link can be made between two devices on the bus. The two devices communicate using basic requests, answers to those requests, and interrupts. The speed at which the devices can communicate is largely determined by the amount of lanes between the devices. PCIe supports multiple lanes, each lane consisting of two serial differential paired wires between the devices. Several types of devices can exist on the PCIe bus. The most common is the endpoint, which exposes its main functionality through PCIe, and mostly completes requests by some other device. Switches provide a way to connect more than two devices together and is capable of routing packets to a correct destination. Bridges can translate PCIe to a different protocol. The root complex (See Section 3.4) is the device at the center of the architecture and provides the connection to the CPU, but can also contain information about connected devices in the system. In newer architectures, the root complex is integrated with the host bridge, the connection to the CPU. The connection to the CPU does not necessarily need to be a root complex, other devices in the system can take that role too. An example topology by PCI-SIG can be seen in Figure 3.1. In Figure 3.3, the PCIe bus is simplified to just the root complex, a switch, and the PCIe device. It is important to understand that while PCIe is called a bus, it shares more likeness with common network protocols.

PCIe has several versions, but they can be backwards compatible with previous versions, even PCI. However, there is no guarantee that a newer function will work with older versions. For example: A device supporting a 5.0 GT/s transfer speed may not support a lower transfer speed of 2.5 GT/s. When first connecting, both parties negotiate the protocol they will use. The pinout of a PCIe slot with one lane can be seen in Table 3.1. The signals indicated with blue are the differential signal pairs. These are used for sending data and for the reference clock. The orange signals are used to help manage the link. The red and green signals are for the JTAG and SMBus signals respectively, which are also present on the PCIe interface.

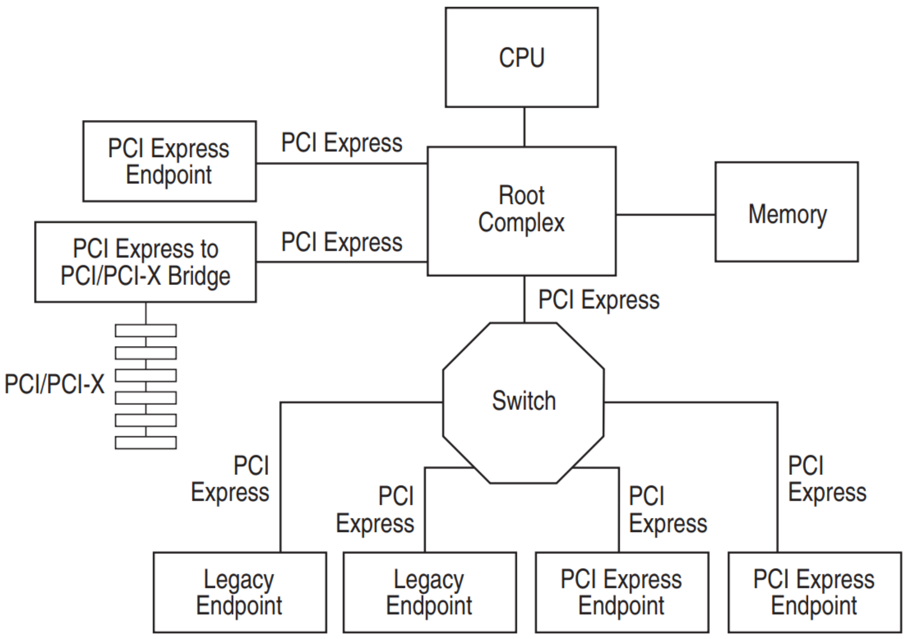


Figure 3.1: PCIe Example Topology from PCI-SIG [3]

3.2. Protocol Layers

The PCIe protocol is divided into three separate layers, which can be seen in Figure 3.2.

- **Physical Layer:** This layer contains the Electrical and Logical sub-blocks and provides the base for simple data transmission.
- **Data Link Layer:** This intermediate layer provides functionality like error detecting/reporting to support the Transaction Layer.
- **Transaction Layer:** This layer contains the main source of communication: the Transaction Layer Packet (TLP). TLPs are used for all high-level communication with a PCIe peripheral

3.2.1. Physical Layer

The electrical part of the physical layer only concerns translating logic into electrical signals and vice versa. The main functionality provided by the physical layer is implemented within the logical block. The data transmitted by PCIe is encoded using symbols. Depending on the transfer speed, this is either 8b/10b or 128b/130b encoding. This means that 10 bits are used to send 8 bits of data. Some symbols or symbol combinations can represent data, others can represent the beginning or ending of packets sent on other layers. If more than one lane is available, the data is interleaved over the lanes, in order to increase bandwidth and prevent different packets arriving at the same time. All control and interrupt messages also use the same data lanes. The data symbols can also be scrambled using a pseudo-random LFSR. This scrambling is turned on by default, but can be turned off during initialization.

The physical Layer also implements the the Link Training and Status State Machine, which is further discussed in Section 3.3, about device initialization. The LTSSM helps in discovering the link width (the amount of lanes which are available), the data rate (the transfer speed at which data can be send) and provides functionality like lane reversal (which can reverse the polarity of lanes).

Pin	Side B Connector		Side A Connector	
#	Name	Description	Name	Description
1	+12v	+12 volt power	PRSNT#1	Hot plug presence detect
2	+12v	+12 volt power	+12v	+12 volt power
3	+12v	+12 volt power	+12v	+12 volt power
4	GND	Ground	GND	Ground
5	SMCLK	SMBus clock	JTAG2	TCK
6	SMDAT	SMBus data	JTAG3	TDI
7	GND	Ground	JTAG4	TDO
8	+3.3v	+3.3 volt power	JTAG5	TMS
9	JTAG1	+TRST#	+3.3v	+3.3 volt power
10	3.3Vaux	3.3v volt power	+3.3v	+3.3 volt power
11	WAKE#	Link Reactivation	PERST#	PCI-Express Reset signal
	Mechanical Key			
12	RSVD	Reserved	GND	Ground
13	GND	Ground	REFCLK+	Reference Clock
14	HSOp(0)	Transmitter Lane 0,	REFCLK-	Differential pair
15	HSOn(0)	Differential pair	GND	Ground
16	GND	Ground	HSIp(0)	Receiver Lane 0,
17	PRSNT#2	Hotplug detect	HSIn(0)	Differential pair
18	GND	Ground	GND	Ground

Table 3.1: Pinout of PCIe slot with one lane (Adapted from pinouts.ru [52])

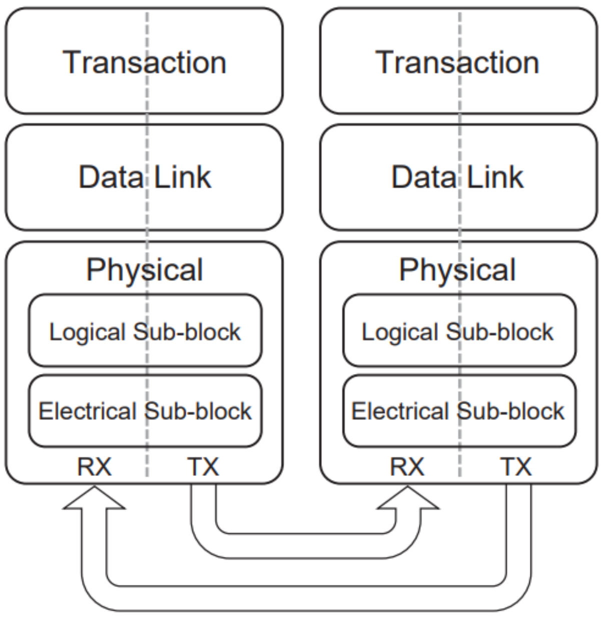


Figure 3.2: PCIe Protocol Layers from PCI-SIG [3]

3.2.2. Data Link Layer

This layer mainly concerns the correct delivery of packets. This is perfomed using acknowledgment packets or ACK, which confirm that a packet has arrived at its destination correctly. The exact opposite, NACK, confirms that there was something wrong with the packet. This allows the sender and receiver to make sure every packet is sent correctly.

In order to not send more packets than the link is physically capable of, the receiver gives credits to a sender, with every sent packet consuming a credit. If credits run out, the sender cannot send more and has to wait for the receiver to give it more credits. This credit system is handled using Data Link Layer Packets, or DLLPs. The data link layer does not check for credits itself. The actual implementation of the credit system is present within in the Transaction Layer.

The data link layer also provides error detection through the Cyclic Redundancy Checks (CRCs) present in the TLPs. In case a TLP arrive with an error, the data link layer provides TLP buffering thereby allowing the TLP to be re-send.

3.2.3. Transaction Layer

This layer creates and controls the Transaction Layer Packets (TLP). It also maintains the credit system mentioned in the data link layer. The TLP consists of a header followed by the data. The header contains relevant information about the destination, receiver, and the contents of the packet. Commonly used TLPs are:

- **Configuration Read/Write** which allows for accessing the configuration space registers, which is discussed in Section 3.3.2.
- **Memory Read/Write** which allows a device to read and write to the main memory. These packets are discussed in detail in Section 3.5. They provide the backbone for DMA transfers over PCIe.
- **Completions** (with or without data). A response packet confirming the (in)completion of an action requested in a received TLP. These packets are also discussed in Section 3.5.
- **Messages** which can contain a wide variety of information like power management, interrupts, vendor defined messages, etc.

3.3. PCIe Peripherals

A PCIe device is a device that is connected to the computer system over a PCIe enabled bus. The connection itself does not necessarily have to be PCIe, as long as it does enable the PCIe functionality. Examples are Intel's Thunderbolt [23], which was mentioned earlier, or DMI [21], Intel's internal bus for communication between the CPU and the southbridge.

3.3.1. Device Initialization

For a new device to start connecting with other devices on a PCIe bus, it first needs to start a process called ”Link Training”. This process starts by synchronizing the clock, using standardized packets. Afterwards, protocol specifics like the data rate or lane width are negotiated, after which the device is able to communicate over the bus. This linking process happens between the device, and the nearest upstream device. This process is handled by the LTSSM (Link Training Status State Machine). After the link training is completed, the device is able to talk on the bus using the correct protocol. This process is also indicated in Figure 3.3. Exact information about the LTSSM can be found in the PCIe specification [3].

Since the device maintains the current state of the link using the LTSSM, checking the status of the LTSSM is possible on that device only. When the device is used in a ”normal” fashion, the device always reaches a resting state in which the device is ready to receive/send packets. The LTSSM is implemented solely on the physical layer. The result of the LTSSM can be used by other layers to confirm the proper functioning of the physical layer.

Afterwards, a second phase called ”Enumeration” is started. Here, the device is given a unique identifier, such that the device knows where in the network it is located. Using this ID, the device can communicate to any other upstream PCIe device, and receive packets from them. The unique identifier consists of three parts:

- **Bus Number:** A bus gets a new number after the downstream port of a switch or bridge.

- **Device Number:** In order to indicate multiple devices on the same bus.
- **Function Number:** The same device can hold multiple functions, which can be addressed differently.

This identifier is referred to as a PCIe ID or BDF (From Bus/Device/Function). Enumeration is an exhaustive breadth first search which requests the first configuration register of a device, containing the Vendor and Device ID. Is there a device on that PCIe ID, the device responds, otherwise the upstream device responds with `0xFFFF`. Based on the Vendor and Device ID, software is then able to find an appropriate driver and initialize the device properly on the software side. Should the device ID and Vendor ID be configured as `0xFFFF`, the software cannot distinguish the response from a default response, therefore not initializing the device at all.

Enumeration is started when the system boots, or when the OS specifically re-scans the PCIe bus. On Linux this can be performed by writing to *"sys/bus/pci/rescan"*, which triggers the enumeration sequence performed by the kernel. It usually happens simultaneously with the MMIO and allocating other memory needed for correct functioning of the device within the OS, as well as finding a correct driver for the device based on the Vendor and Device ID of the device. This process is indicated with a green arrow in Figure 3.3. The BIOS can also start the enumeration process early during boot. The MMIO of the root complex is usually part of the stolen memory space and cannot be accessed later. On Linux, a manual rescan of the PCIe bus, also triggers enumeration of any new devices found on the bus. Through placing debug information within the kernel, it is possible to see exactly the interaction the kernel has using PCIe with a new device during the enumeration sequence. A short excerpt of the interaction can be seen in Appendix A.

3.3.2. Device Configuration

The general way of configuring a PCIe device is through the registers on the device. MMIO makes these registers more accessible to the main system, as previously mentioned in Section 2.3.2. The main registers can be found in the Configuration Space, which is a standardized set of registers. The configuration space contains information like the Device and Vendor ID, based upon which an OS can load a correct driver, or readstatus registers, with which the device can announce its status to the OS or driver. These registers can also be written to by using TLPs. Examples are the interrupt register, or registers that control options like time-outs or link speed. For more, and device specific, registers, the Base Address Registers (BAR) are used. The standard configuration space for PCIe endpoints contains two BARs, but more are possible.

The device can also use Extended Capability registers to advertise device capabilities. These registers form a linked list between their headers, in which the header contains a pointer to the next header, for the next capability. These capabilities can be used by software to improve interactions with the device. Examples are Advanced Error Reporting (which extends the errors the device can report, for better debugging) or Address Translation Services (which allows the device to perform Virtual Address Translations itself).

3.3.3. Hot-plug

It is possible to hot-plug a PCIe device as per the PCIe specification. This requires support from three different parts of the system: the software, the PCIe slot, and the PCIe device. The core principle of hot-plug according to the specification provided by PCI-SIG is to have both the slot and the device keep the software updated on the current status of the device. This way, the software can anticipate removal and/or addition of devices and provide feedback to the device and slot on how to do this correctly.

The "Slot Capabilities" Register is a standard register, which can provide information about the slot, including Hot-plug support. This register is implemented in the switch that provides the slot. Hot-plug support in software and devices is common, but Hot-plug support in slots is not. The two servers that were tested did not have Hot-plug capable slots. This may change in the future due to the rise of PCIe capable user-interfaces like Thunderbolt.

Hot-plug support can be divided into two sections: electrical (involving the device and the slot/-motherboard) and software (which involves the slot and the OS). If an electrical Hot-plug succeeds, the software may be informed of the device anyways (by for example a manual rescan). A succesful electrical hot-plug would mean the device is successfully connected to the motherboard and has completed the LTSSM. A successful software hot-plug would mean enumeration of the device and that software initialization was successful.

If an electrical Hot-plug is not supported, the system may respond unexpectedly to removing or adding new devices. On some systems, plugging in a new device resulted in a complete and immediate system failure and shutdown. In other systems, the operating system started to lag and slow down, or the operating system did not respond and continued to operate normally.

3.4. Root Complex

The root complex is the top-level entity within the PCIe hierarchy. It always has a PCIe ID of `0x0000` and its main functionality is providing the interaction with the memory and is usually combined with being the host bridge, i.e., the bridge between the processor and the PCIe bus. During boot, the BIOS or OS configures the root complex using the configuration registers.

The root complex is often located in the CPU and can also be seen in Figure 2.6, where it is located on the DMI bus (the internal bus between the southbridge (PCH) and the CPU) and is depicted as the ”Host Bridge”. It can also be seen in Figure 2.7, where it is part of the DMA process. It is from the root complex where the enumeration process is started. The BIOS or OS utilizes the root complex to send and receive the packets on the PCIe bus required for enumeration.

The root complex also provides the interaction with memory. DMA using PCIe uses the packets described in Section 3.5. These packets are send to the root complex and the root complex also sends back the required Completion packets.

3.5. DMA Using PCIe

This section discusses the implementation of DMA on PCIe. For a general explanation of DMA, more information can be found in Section 2.5.2. When a device is ready to read or write to the main memory of the system, a PCIe peripheral can send a read or write memory TLP on the PCIe bus, which is automatically routed to the root complex. The data from the memory is used by the root complex to route a Completion packet back based on a requester ID send with the request TLP. This requester ID is the PCIe ID acquired during enumeration. This means a device that does not have a PCIe ID (e.g. because it is not enumerated) can (in theory) not perform DMA.

DMA using PCIe is therefore mainly performed using TLP’s on the PCIe Bus. The two main TLPs that are being send back and forth during a DMA request are MRd (Memory Read) and Cpl/CplD (Completion/Completion+Data). The purpose and headers are explained below:

Memory Read Requests There are two types of MRd TLPs: MRd32 and MRd64. The difference is the length of the address that the TLP contains and a change in the fmt space in the first bits of the TLP, indicating the length of the header. Some protocols can enable PCIe, but don’t enable 64-bit addressing. The MRd64 header format can be seen in Figure 3.4.

The TLP consists of the following fields:

- **Fmt + Type** These indicate the type of TLP that is being sent. For Mrd64 this would be `0x010000`
- **Traffic Class (TC)** The traffic class of the TLP. This may change the way the TLP is routed.

Each TC may belong to a Virtual Channel for higher priority routing, or another different routing

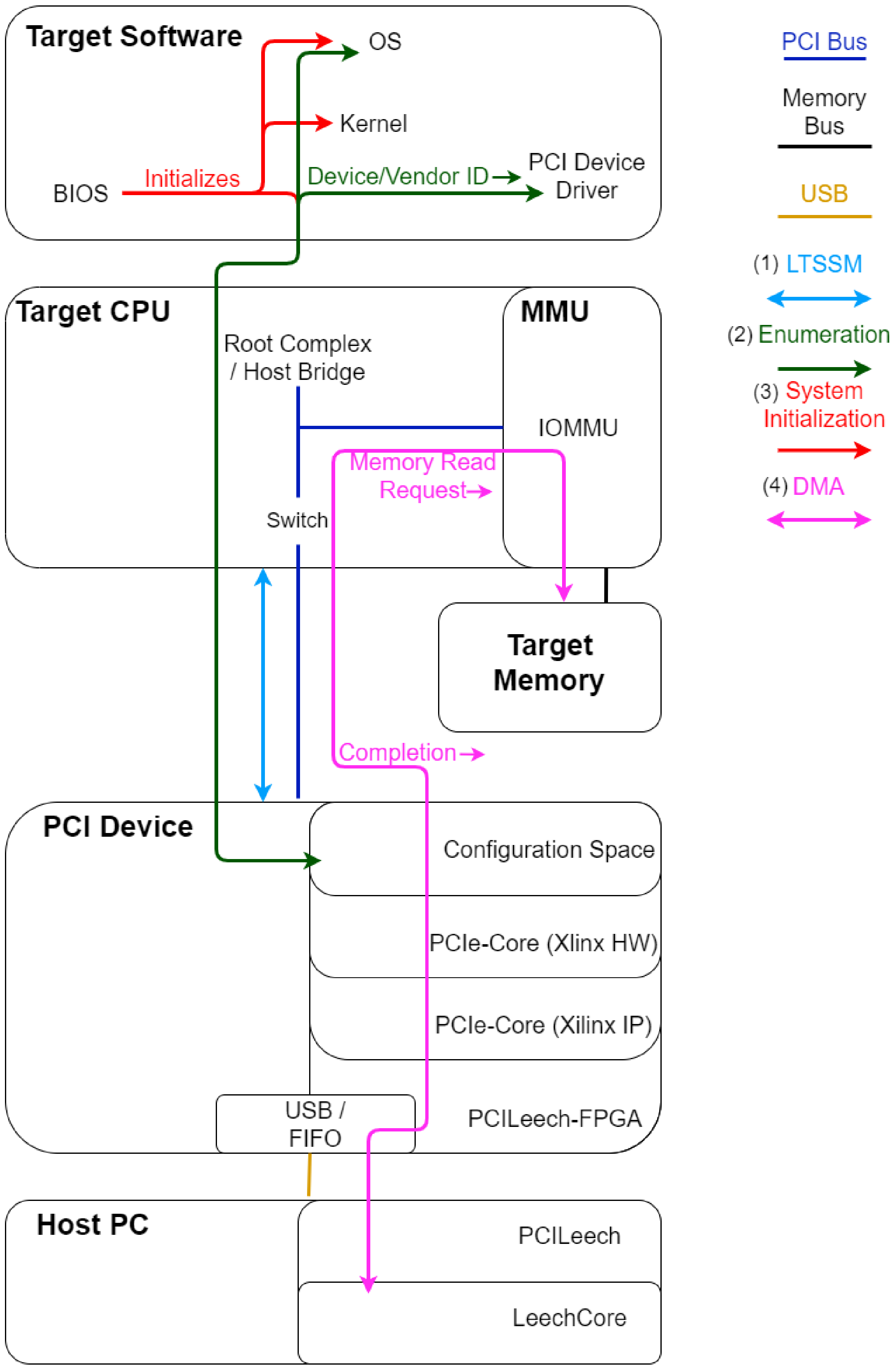


Figure 3.3: Full overview of several mechanics of PCIe and DMA

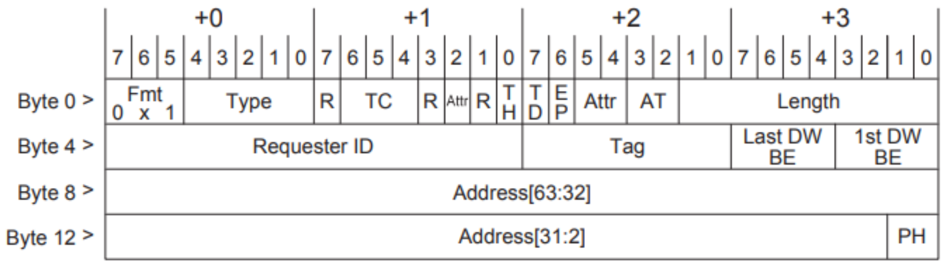


Figure 3.4: MRd64 header format [50]

treatment.

- **Attributes** Several attributes to configure the ordering of the request. Also contains a ”no snoop” preference, which is used for hardware coherency management and can prevent other devices from interpreting (snooping) the packet.
- **Trafficking and Processing Hints (TH & PH)** TH indicates the presence of processing hints (in PH). PHs may be used to help the memory completer with information about the frequency and temporal locality of the memory request.
- **TLP Digest** TD indicates the presence of a TLP digest. For example, using ECRC to validate the integrity of the TLP.
- **Error Forwarding / Data poisoning** When a packet is malformed, it may be useful to still route the packet to the destination. In that case, the EP bit is set in order to confirm a malformed packet.
- **Address Translation** Used to indicate whether the address in the TLP is translated or not. Used for ATS.
- **Length** Used to indicate the length of the requested memory.
- **Requester ID** The source PCI ID of the requester. Completions are routed back to this address.
- **Tag**
  A tag used for tagging transactions. A completion will return with the same tag, allowing the requester to link a request with a completion.
- **DW Enable**
  A mask for which Double Word should be read out (or written with a MWr64). Used for fine-grained control over reads or writes.
- **Address**
  The actual base address of the memory request.

Memory Read Completion There are two types of Completion TLPs: Cpl and CplD. The difference is the second one contains data and a change in the length field, according to the amount of data send back. The Cpl Header can be seen in Figure 3.5.

Most fields in the Cpl TLP are similar to the one in a MRd TLP. In case of a CplD TLP, the data is send directly after the header. The Cpl TLP contains the following different fields:

- **Completer ID**
  The Device that completed the transaction. For MRd requests, the completer is the root complex, meaning this field will be set to `0x0000`.
- **Completion Status**
  Indicates the status of the completion. With no error, this is set to all 0. For an unsupported request, for example when attempting to access inaccessible memory space, the status is set to 1.

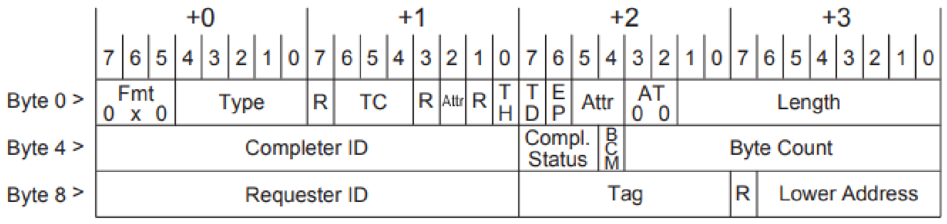


Figure 3.5: Cpl header format [50]

- **Byte Count Modified**
  Used only by PCI-X Completers.

4
Memory Acquisition Techniques

This chapter discusses the different concepts and techniques for acquiring memory. They are classified based on the interface at which the acquisition happens in Section 4.1. These classes are software based techniques (Section 4.2), DMA based techniques (Section 4.3), and DRAM based techniques (Section 4.4). The target scenario in which these techniques would be required to function is discussed in Section 4.5 Next, a model is used to determine to most effective method of memory acquisition. This chapter discusses the reasoning behind the model in Section 4.6, applies it to available hardware methods in Section 4.7, and discusses the result in Section 4.8.

4.1. Classification of Memory Acquisition Techniques

The different acquisition techniques are divided into three categories. The first two categories are based on the two main memory access processes: through the OS, or through DMA. The third type attempts to directly connect to the DRAM or its interface: the memory bus. Software based techniques require some sort of program running on the CPU itself. They use the regular means of accessing the DRAM, which is through the MMU. DMA based techniques use any type of interface that supports DMA. If the host directly connects to such an interface, it is classified as a direct DMA based technique. If a device which is already connected to such an interface is targeted, then it is considered an indirect DMA based technique. The last technique targets the DRAM in a more direct way. This concerns attacks that focus on the DRAM-stick itself or the DDR bus with which the DRAM is connected to the CPU.

4.2. Software Based Acquisition Techniques

The first type of acquisition technique is software based. This includes running native or 3rd party software tools on the system itself to provide a memory snapshot.

- **Virtual Machine**
  In some cases the server is not running directly on machine hardware, but on a virtual machine. In that case, the virtual machine can provide native tools to directly store and access a memory snapshot. Virtualbox is an example of a Virtual Machine providing that functionality [28]. Although native tools like this are fast and reliable, they require admin access and a login, and it only works if the target system is run on a Virtual Machine.
- **Native OS Tools**
  Windows provides a native toolset for making memory images and analyzing them [14]. Although this is fast and reliable, it requires admin access, a login, and a reboot. It also leaves a memory footprint (overwriting part of the memory snapshot), and the source code of the application is unknown, making it difficult to verify the correct functionality of the application.
- **Hibernation File Conversion**
  Windows stores a hibernation file on your hard drive to reconstruct the system state from after

hibernation. This hibernation file can be unpacked into a readable format by a 3rd party tool [58]. In contrast to the previous method, this method does not leave a memory footprint. However, it still requires a reboot to generate the hibernation file and admin access and a login to access the file. The extent to which parts of memory are stored in the hibernation file is also unknown.

- **3rd Party Tools**
  There are various 3rd party software tools on the market that can provide memory imaging. A collection can be found in Table 4.1. On Windows, these tools usually do not always require admin access, but this is a requirement on Mac OS and Linux. These tools are usually fast and do not require a system reboot, but still require a login and leave a memory footprint.

- **Warm Reboot**
  It is sometimes possible to reboot the system without losing power to your memory. If the BIOS is also hijacked, it is possible to boot a new OS, specifically designed for memory acquisition [56]. Although this methods circumvents the need for admin access or a login, it still leaves a memory footprint. The data collection is performed over a serial bus, resulting in a very slow acquisition.

Windows	winpmem, DumpIt, Magnet RAM Imager, Memoryze,
	FastDump, BelkalMager, RamCapture
Linux	Linpmem, LiME
Mac OS	Osxpmem, Memoryze

Table 4.1: Different 3rd Party Tools for RAM Imaging

4.3. DMA Based Acquisition Techniques

DMA provides a straightforward way of accessing the memory. This category is divided into two subcategories: if the DMA channel can be directly accessed by a device the attacker chooses, it is classified as a direct DMA channel. If it requires subverting an existing device with a DMA channel, it is classified as an indirect DMA channel.

4.3.1. Direct DMA Channels

- **FireWire**
  Firewire, also known as IEEE 1394 [2], is an old protocol used for high-speed data communication which also supports DMA. It is fast, reliable, does not need login or admin access, and is hot-pluggable. FireWire can be utilized easily to acquire memory [32]. FireWire slowly lost support in the computing world and it is not regularly used in servers any longer.

- **PCIe**
  PCIe is another protocol capable of making DMA requests. By inserting a device that can access the PCIe bus, it is possible to make DMA requests and acquire memory through the PCIe bus [27]. This requires a host computer to control the device through software [30]. Although this method is fast and widely supported, it suffers from stability issues and relies on the host PC properly accepting the PCIe device before a DMA call can be made, sometimes requiring a reboot before it accepts the device. It is also countered by the IOMMU, which can block memory accesses from PCIe devices.

- **Thunderbolt**
  Thunderbolt [23] is a relatively new protocol and can be run over USB. Thunderclap [59] is an attack that uses this protocol to access the memory. When the vulnerabilities of Thunderbolt were revealed, improvements were made to limit the capabilities of Thunderbolt, but those improvements were circumvented as well by ThunderSpy [55] in 2020. Thunderbolt is not featured on many servers yet, but can be a fast and stable method of memory acquisition. Any acquisition method using Thunderbolt would need to work around the newer security measures, as well as the IOMMU.

4.3.2. Indirect DMA Channels

Since devices over direct DMA channels struggle to gain an accepted or trusted position in the system, one could also look for devices that already have a trusted position within the system.

- **Rogue Devices**
  A common device in every server with DMA access over the PCIe bus is the Network Interface Controller (NIC). The controller can be compromised and its firmware replaced [45]. This method is fast and does not leave a memory footprint, but NIC's still vary a lot per server. This means it is difficult to develop broad support for this technique.

- **LOM**
  Most servers also carry some form of Lights-Out Management like integrated Lights-Out for HPE and Remote Access Control for Dell. These provide some form of out-of-band management and implement common protocols such as IPMI. LOM's have been generally considered to be insecure [25] [16]. This makes it possible to find an exploit, patch new firmware and expose DMA requests [51]. However, finding an exploit can take time and existing exploits can be patched. On top of that, different manufacturers have different implementations, which makes it difficult to develop broad support for this technique. The advantage of this method is that the LOM generally is a high-level entity within the system, even hidden to the real OS.

4.4. DRAM Based Acquisition Techniques

The memory can also be accessed directly over the memory bus and also can be targeted directly, which constitute the last category of memory acquisition method. In contrast to software-based memory acquisition techniques, hardware-based techniques have the advantage that they do not require admin access or a login.

- **Cold Boot**
  As discussed in Chapter 2, once the DRAM loses power, it loses its data because the volatile RAM needs to be refreshed. By cooling down the memory chip, the charge that maintains the bit's value depletes slower. By taking the memory out of the target system and transplanting it into a controlled system, the memory can be read out [63]. This method is fast, but the charge decays quickly and requires a complex setup. It is also countered by the Intel Memory scrambler (see Section 2.5), which would need to be reverse-engineered for the method to work correctly.

- **MMU Impersonation**
  It is possible to connect another MMU directly to the memory chip and communicate with the memory through that MMU instead [9]. This method is complex due to the signal integrity issues already existing in the DDR protocol and the fact that there is already an MMU connected to the memory. It is also directly countered by the Intel Memory scrambler, which needs to be reverse-engineered for the method to work correctly.

- **Floating Gate SEM analysis**
  Reading out an EEPROM using an electron microscope is possible [24]. This directly reveals the information stored on the chip. Since DRAM also consists of a floating gate transistor based memory cell, a similar technique might be applied in tis case. This technique would also be disadvantaged by the fast-depleting capacitance, so cooling might me needed. It also requires disassembling of the RAM and is also countered by the Intel Memory Scrambler.

4.5. Target Scenario

In order to limit the scope of the thesis, a scenario in which a solution would need to operate is described below.

- This thesis focusses specifically on one computer system: a rack server. This is a type of computer system specifically build to perform high-end computing tasks and fitted to a flat form factor to optimize the use of physical space.

- The main advantage in the scenario described above is that physical access to the server is considered to be available. This grants direct access to the hardware and software on which the system runs, opening up several memory acquisition techniques.
- we consider the server running the latest version of Linux OS with all patches and updates from the server manufacturer applied. Linux was chosen because NFI was interested in this OS specifically and Linux is easier to debug than other OSs.
- We consider the operating system to be locked and no login credentials are available to unlock the operating system, the BIOS, or the LOM.
- The system is turned on. We assume that a reboot can lead to a loss of information in the memory. Therefore, a reboot must be avoided.
- The system does not have an active IOMMU (See Section 2.4.2). The IOMMU and its vulnerabilities are discussed in Section 5.3.2, but a workaround / solution is not expected or necessary.

This scenario immediately excludes most software based techniques, since they require login credentials. This leaves the following techniques to be evaluated:

- Warm Reboot
- DMA using FireWire
- DMA using PCIe
- DMA using Thunderbolt
- DMA using a Rogue Device
- DMA using the LOM
- Cold Boot
- MMU Impersonation
- Floating Gate SEM analysis

4.6. Model Creation

The model is based on the Attack Vector Evaluation Model by van Beusekom [62], which was in turn based on the CVSS3 Vulnerability Assessment Model [4]. The model was altered to only use positive metrics and the descriptions of certain metrics was changed to better fit the the use-case of memory acquisition. The different metrics are described and divided into four categories. A low point value is considered favorable to a high point value.

4.6.1. Research Complexity

Full point values in Table 4.2

- **Knowhow:** The amount of missing knowledge, before work on development of the method can start.
- **Effort:** The amount of setup/research time required to develop the method. This includes the development of workarounds to fit the scenario described in Section 4.5.

4.6.2. Execution Complexity

Full point values in Table 4.3

- **Specialized Tools:** To what extent the method needs any specialized tools, or knowledge about specialized equipment to execute the method.
- **Time Requirements:** How long the acquisition takes to execute. Infeasible if the acquisition takes longer than the time this thesis takes to write. A memory acquisition that takes too long is also undesirable with regards to forensic soundness of the acquired image. The memory image acquired should resemble the memory image at device seizure as closely as possible and with a long acquisition time, the confidence in a correct memory image is decreased.

4.6.3. Usefulness

Full point values in Table 4.4

- **Reproducibility (same device):** The time it takes to reproduce the same attack on the same device.
- **Reproducibility (similar device):** The time it takes to reproduce the same attack on a similar device.
- **Determinism:** How often a properly executed attack will yield results. The attack should produce consistent and correct results in order to be used in forensic investigations in the field.
- **Reusability:** To what extent the attack can be reused among different manufacturers/devices/systems.
- **Device Impact:** To what extent the attack invalidates the security for the entire device or parts of the device.

4.6.4. Risk

Full point values in Table 4.5. The original model included economic and environmental Risk. These have been left out of this model since they are not relevant to these methods.

- **Personal Impact:** To what extent the attack impacts areas such as personal privacy, security or freedom.
- **Destructiveness:** To what extent the attack destroys the targeted device. Risk of destruction of the device also increases the risk of loss of information stored on the device.

Research Complexity Knowhow	0	5	10	20	50
	No knowledge or outstanding assumptions required	Missing knowledge or outstanding assumptions require little time investment to resolve	Missing knowledge or outstanding assumptions require considerable time investment to resolve	Missing knowledge or outstanding assumptions require significant time investment to resolve	Missing knowledge or outstanding assumptions require extreme time investment to resolve
Research Complexity Effort	0	5	10	20	50
	Virtually no setup/research time	Some setup/research time	Considerable setup/research time	Significant setup/research time	Extreme setup/research time

Table 4.2: Elaborate Values for Research Complexity Metrics

Execution Complexity Specialized Tools	0	5	10	20	50
	Requires no specialized tools or knowledge	Cheap specialized tools and limited specialized knowledge required	Expensive specialized tools and knowledge required	Very expensive specialized tools and knowledge required	Extremely expensive specialized tools and knowledge required
Execution Complexity Time Requirements	0	5	10	20	50
	Attack takes up to a few seconds	Attack takes up to a few minutes	Attack takes up to a few hours	Attack takes up to a few days	Attack takes longer than a few days

Table 4.3: Elaborate Values for Execution Complexity Metrics

Usefulness Same device reproducibility	0 Attack can be reproduced on the same device without extra research	5 Attack can be reproduced on the same device with minimal extra research	10 Attack can be reproduced on the same device with moderate extra research	20 Attack can be reproduced on the same device with extensive extra research	50 Attack can be reproduced at the same time it took to attack the first device
Usefulness Similar device reproducibility	0 Attack can be reproduced on a similar device without extra research	5 Attack can be reproduced on a similar device with minimal extra research	10 Attack can be reproduced on a similar device with moderate extra research	20 Attack can be reproduced on a similar device with extensive extra research	50 Attack can be reproduced at the same time it took to attack the first device
Usefulness Determinism	0 Properly executed attack will always yield results	5 Properly executed attack will generally yield results	10 Properly executed attack will mostly yield results	20 Properly executed attack will sometimes yield results	50 Properly executed attack will rarely yield results
Usefulness Reusability	0 Results can be reused for all computer systems	5 Results can be reused for all server manufacturers	10 Results can be used among device family	20 Results can be used among production batch	50 Results are per device and cannot be carried over
Usefulness Device Impact	0 Attack results in total loss of security of entire device	5 Attack results in total loss of security for important subsystems, but not entire device	10 Attack results in partial loss of the device's security	20 Attack results in partial loss of the device's subsystems	50 Attack does not result in (additional) loss of security of the device or its subsystems

Table 4.4: Elaborate Values for Execution Usefulness Metrics

Risk Personal Impact	0 No impact in areas such as personal privacy, security, or freedom	5 Slight impact in areas such as personal privacy, security, or freedom	10 Considerable impact in areas such as personal privacy, security, or freedom	20 Significant impact in areas such as personal privacy, security, or freedom	50 Severe impact in areas such as personal privacy, security, or freedom
Risk Destructiveness	0 No risk for the device	5 Low risk to device, opening the casing, soldering on the device required	10 Medium risk to the device, Desoldering or transplanting required	20 Higher risk to device, attack may damage device	50 High risk to device, attack may destroy device completely

Table 4.5: Elaborate Values for Execution Risk Metrics

4.7. Model Application

This section discusses the reasoning behind the score of the methods per model metric. The precise scoring can be found in Table 4.6.

- **Research Complexity - Knowhow:**
  Memory acquisition techniques using a DMA enabled port are less complex and better documented than the other techniques. A warm reboot based method is also executed using simple(r) protocols. Using a rogue device or a LOM requires extra research into the vulnerable piece of the target device. Research into Cold Boot, MMU impersonation, or SEM analysis requires more research into the physical properties of the memory.

- **Research Complexity - Effort:**
  RAM acquisition methods over DMA enabled ports are already documented and proven to be viable. However, ports like FireWire and Thunderbolt are rarely present on server systems (See Appendix D). For warm reboot, some more research needs to be done for it to work on a broad spectrum of targets. For indirect DMA attacks, it is usually necessary to find an exploit, which would be difficult. Researching DIMM level attacks would take a lot of time to both reverse-engineer the Memory Scrambler as well as deal with the signal integrity issues. SEM analysis on DRAM would most likely require extreme amounts of effort to get working.

- **Execution Complexity - Specialized Tools:**
  FireWire and Thunderbolt are protocols intended to be directly accessed by consumers. This means that there will be very little speicialized tools required to target those channels. PCIe and Warm Reboot require additional tools, but those are readily available. A Cold Boot setup is difficult because of the temperature requirements. A MMU impersonation is difficult due to the engineering issues with regards to signal integrity. SEM analysis requires a SEM, which is an expensive piece of hardware.

- **Execution Complexity - Time Requirements:**
  Methods with a connection over a PCIe enabled bus are fairly fast and the attack takes up a few minutes. The Warm Reboot is over a Serial connection and is therefore slower. The bottleneck for the LOM is the connection to the LOM itself, which is most likely over TCP. Even if it is possible to read a single bit with SEM analysis, reading out the entire memory will take a long time.

- **Usefulness - Reproducibility (Same Device):**
  If the attack works once for the same device, there is no reason why it should not work a second time. The DIMM level methods might deal with an LFSR, which may cause some extra time.

- **Usefulness - Reproducibility (Similar Device):**
  If the attack works on one device, it should generally work for a similar device with minimal effort. Exceptions to this are the Rogue Device and LOM methods, since there is no guarantee that the similar device has the same hardware and software available that you targeted, which means a new plan of attack needs to be made for the new device, requiring more extensive research. The viability of SEM analysis needs to be evaluated per device, since different memory may have a different layout.

- **Usefulness - Determinism:**
  A properly executed attack should yield results in most cases. The DIMM level methods, which may struggle with decrypting the relevant data because of the memory scrambler. The viability of SEM analysis on DRAM is unproven.

- **Usefulness - Reusability:**
  The only interfaces with memory access consistently present on all servers are the PCIe and DIMM interfaces. The warm reboot attack may struggle to find the same or similar patterns of connections. The rest of the attack interfaces are only sparsely available.

- **Usefulness - Device Impact:**
  The LOM is a high level entity, with access to almost anything an OS has. If a technique results in full live memory access, it is most likely also able to subvert the OS. Warm reboot, cold boot, SEM analysis only target the RAM and disable the system completely.

- **Risk - Personal:**
  All methods cause the same level of privacy violation, since they all have the same end result. The goal of the attack is to achieve a privacy violation: the memory contains private data. It is preferable to have an technique to be able to only target the memory data needed. For example: a server with four users, of which only one is malicious. A method that could focus its memory acquisition only on the malicious user is preferable over a method that collects the entire memory regardless. That being said, determining the ”correct” memory that needs to be acquired can lead to (possibly) missing valuable data. Whether or not data is ”valuable” can always be determined after the acquisition. Therefore, although DMA-enabled methods can target memory ranges seperately during the acquisition, all methods are considered to have a significant impact on personal privacy.

- **Risk - Destructiveness:**
  Thunderbolt, FireWire, and the LOM are/have user-level interfaces and do not require the device to be opened. Warm Reboot, PCIe, Rogue Devices, LOM, and MMU impersonation require the device to be opened to access the relevant interface. A Cold Boot attack also requires transplanting the DIMM’s, resulting in a larger risk to the device.

4.8. Results

The results can be seen at the top of Table 4.6. The lowest score indicates the most effective memory acquisition method, which is DMA using PCIe. DMA attacks in general provide a fast way to access the memory, while not requiring as much knowhow, since they are well researched. Specifically DMA using PCIe is a relatively low effort attack since the equipment and effort to perform the basics of the attack are already available. This thesis will continue to look into how DMA using PCIe functions, and to what extent the technique can be improved upon.

	Relevant Attacks:	Warm Reboot	FireWire	PCIe	Thunderbolt	Rogue Devices	LOM	Cold Boot	MMU Impersonation	SEM analysis
	Total Score:	90	140	65	145	170	145	205	200	440
Research Complexity	Knowhow	10	5	5	5	10	10	20	20	20
	Effort	10	50	10	50	20	20	50	50	50
Execution Complexity	Specialized Tools	5	0	5	0	0	0	20	20	50
	Time Requirements	10	5	5	5	5	10	5	5	50
Usefulness	Reproducibility (Same Device)	0	0	0	0	0	0	5	5	0
	Reproducibility (Similar Device)	5	5	5	5	50	30	5	5	50
	Determinism	0	0	0	0	0	0	20	20	50
	Reusability	20	50	0	50	50	50	0	0	50
	Device Impact	5	10	10	10	10	5	50	50	50
Risk	Personal	20	20	20	20	20	20	20	20	20
	Destructiveness	5	0	5	0	5	0	10	5	50

Table 4.6: Model Score for RAM Acquisition Methods per Metric

5 DMA-Based Memory Acquisition Using PCIe

This chapter further details the concept of what a DMA attack exactly is. It first discusses existing DMA attacks in Section 5.1 and afterwards it explains the main countermeasures and drawbacks of acquiring memory based on these attacks in Section 5.2. Based on a vulnerability analysis of PCIe and DMA in Section 5.3, improvements to the memory acquisition method are proposed in Section 5.4.

5.1. Existing DMA Attacks

DMA Attacks are attacks that abuse DMA in order to gain access to the systems memory, or subvert the system by writing to the memory. The actual ”attack” is not an attack on DMA, but an attack on the system memory or a process within system memory.

The first use cases of DMA-based memory acquisition were not malicious, but instead focussed on memory analysis for forensic or investigative purposes. Dedicated PCI cards like Tribble [17] or CoPilot by Komoku [65] were intended to for discover, detect, and remove memory rootkits. Both these cards worked according to the standard DMA principles. The card is insterted before system boot and used DMA to access the system memory. The memory could be scanned or transferred to a permanent storage device for analysis. Should malicious data be encountered on the memory it could be destroyed by writing to it. Rutkowska [54] predicted multiple countermeasures against this type of memory acquisition and named in particular the IOMMU as the final nail in the coffin for memory acquistion, stating one could ”say goodbye to hardware-based memory acquisition” due to the IOMMU.

The first ”successful” commercial DMA attacks came with Inception [46] in 2012. Inception used the FireWire [2] or IEEE1394 to access system memory. Using an Adapter, Inception could also work on PCI/PCIe ports. Inception provides easy methods to unlock a target system, insert Kernel Module implants, and privilege escalation to further attack a computer system. Inception was not the only tool to use the FireWire port, but was the first one to be widely commercially adopted. FinFireWire [35] by Gammagroup was already used by government agencies and provides more functionality in the form of memory image analysis. The main drawback of using FireWire is its limited to accessing 4GB of memory and the form factor was not widely adopted.

The next main commercially available DMA attack was PCILeech [30]. PCILeech provides similar functionality to Inception, but with a flexible front- and back end. PCILeech supports multiple access devices like FPGAs connected to PCIe or M.2, USB hubs, or compromised HP iLO on the target system. It provides an API which can be utilized to develop exploits on the memory (for unlocking the system or implanting memory), or forensic memory analysis. When using a PCIe card, the process is similar to earlier DMA attacks. The card is inserted to the target PC, which immediately allows for DMA access. Its main drawbacks are the same: The card has to be inserted beforehand, and (if implemented correcly) the IOMMU still counters PCILeech by severely restricting the memory regions it is able to access.

The Thunderbolt [23] port facilitates PCIe and can therefore also be used for DMA. In 2019 Markettos et al. [59] presented ThunderClap, which emphasized the danger of DMA attacks by presenting several attack scenarios and discussing IOMMU vulnerabilities. This work discussed in a more formal fashion the affected (operating) systems and analyzed to what extent a DMA attack can impact it. Their main contribution is the analysis of the shortcomings of current IOMMU implementations. Their platform can attack a system using Thunderbolt or PCIe. The extension of a DMA attack to Thunderbolt is an important development considering the recency and wide adoption of Thunderbolt. ThunderSpy [55] also uses the Thunderbolt port on a Mac to perform a DMA attack. Since MacOS automatically enables an IOMMU, which resides on a separate chip on the motherboard, this attack is usually countered. By flashing new firmware on the chip containing the IOMMU, ThunderSpy is still able to perform DMA over Thunderbolt.

In conclusion, current attacks using DMA all revolve around plugging in a malicious device, or compromising a device that is already plugged in. After that, the device is ready to read or write to the main memory of the system, provided there is no IOMMU. The core of a DMA attack using PCIe are the TLPs presented in Section 3.5, which are used to read memory data.

5.2. DMA Limitations

DMA attacks over PCIe suffer from the following drawbacks or countermeasures.

Unavailable Slots

If all slots on a system are already in use, there is no other options than to un-plug a device, in order to free up a slot. Unplugging a device will change the system state, which may alert the system that something is wrong, makes the memory image less true to the image at device seizure, and may also cause electrical instability of the system. Unplugging a critical device like a storage controller may lead to a full system crash.
As long as a newly plugged in device at the same slot is not enumerated, the device will not respond to requests directed at the PCIe ID belonging to the slot.

Electrical Instability while Hot-Plugging

As described in Section 3.3, Hot-Plugging a PCIe device can lead to system failure. In most crashes, the system indicates a lost connection to other peripherals within the system. The other devices may either be triggered by an electrical glitch caused by the inserted device itself, or an unexpected system response within the OS, CPU, or chipset.

LTSSM interaction while Hot-Plugging

When the device is inserted, it first needs to be able to communicate on the PCIe bus. As described in Section 3.3, this happens through the LTSSM. The LTSSM is an interaction between the upstream and the inserted device. This means that the upstream device still needs to recognize the insertion and proceed with the LTSSM process in order for correct operation of the downstream device.

Enumeration while Hot-Plugging

The lack of enumeration means the device is not able to use its own PCIe ID to send TLPs since it has none. The lack of a PCIe ID means the device can (in theory) not perform DMA.

Software Recognition while Hot-Plugging

The lack of software recognition means the device is not able to be connected to a driver or use its capability registers to advertise device capabilities. A device is only recognized by software during the enumeration process. If the device is enumerated it will have impact on the memory through MMIO (See Section 2.4). This prevents a completely clean memory image.

Rebooting

For forensic research it is important that the acquired memory data represents the memory at the time of the device seizure as closely as possible. A reboot may cause a loss of information, which is why a hot-plugged DMA attack is preferred.

IOMMU

The main countermeasure against DMA attacks is the IOMMU. A correctly implemented IOMMU only allows a peripheral to access the memory using virtual addresses provided by the IOMMU. A further analysis was made in Section 2.4.2.

RAM Encryption

DMA attacks can be countered using RAM encryption [48]. The key to a successful RAM encryption method is the protection of the encryption key. As storing the key in RAM would not work, this means the key is hidden in another I/O device or in seperate registers within the CPU [48]. Once the key is compromised, the encryption becomes useless [15].

5.3. Vulnerability Analysis

Several vulnerabilities still exist within the PCIe protocol and the IOMMU.

5.3.1. PCIe Vulnerabilities

PCIe is a common and unsecured protocol which can access the memory through DMA. Vulnerabilities within PCIe can aid an attacker in executing a successful DMA attack:

- **PCIe ID Spoofing**
  The PCIe ID is used to route packets back to the device where they came from. A PCIe device can also read packets with a different ID, when they are routed on the same or downstream busses. This way, a device can effectively impersonate any other PCIe device. This allows a non-enumerated device to spoof any ID in order to perform DMA requests. A simple countermeasure against ID Spoofing is Access Control Services (ACS), a PCIe Capability implemented in PCIe Switches. ACS can block suspicious or unexpected PCIe traffic. Although ACS is recently gaining more support in Intel processors [70], it is not used a lot.
- **Changing Configuration Space**
  Modifying the configuration space of the device can lead to different interactions with software. Changing the Vendor or Device ID allows the device to pretend to be a completely different device. This changes the way software interacts with the device. For example, it might load a different driver. This could be for example used to load a particularly badly written driver in order to abuse some vulnerability within that driver. The capability registers, which indicate extra device functionality, may cause certain software functionality to be prepared for the device as well. As mentioned earlier in Section 3.3.1, a Vendor and Device ID of `0xffff` can be used during the enumeration process as well. Since this process starts with reading these registers and stops when reading `0xffff` since it is a response that indicates the lack of a device.
- **Lack of ID checking at memory request**
  The PCIe ID passed on with a memory request is not checked. This means non-existing IDs will still trigger a response from the Root Complex.
- **Routability of non-existing PCIe IDs**
  PCIe switches can route PCIe IDs which are non-existent. The exact behaviour is unknown, since this falls outside the specification. A PCIe ID with a very high bus number is likely to pass through the entire system, since all switches will route it through to the next bus if they cannot route it themselves properly.

5.3.2. IOMMU Vulnerabilities

It has been shown that the IOMMU is not bulletproof yet [59]. However, enabling it makes creating a complete memory image very difficult. Vulnerabilities often rely on further escalation in order to provide complete access to the entire memory. This often means completely disabling the IOMMU or bypassing it altogether.

PCIe vulnerabilities still apply when interacting with the IOMMU:

- **PCIe ID Spoofing**
  Unless specifically prevented by ACS, ID spoofing still applies. This means an attacker still has access to all memory spaces allocated to any peripheral. If there is a peripheral capable of passing through the IOMMU, spoofing that ID would most likely allow for full access.

- **Changing Configuration Space** Loading certain drivers that implement certain device transactions badly are still present in Linux. Exploitation of the driver interaction may still lead to memory access [41], but most likely requires some form of escalation. Since the device needs to be recognized by software before the driver is loaded, a reboot is also required.

The IOMMU uses the PCIe ID to provide address translation. This means non-existing IDs are not going to be able to pass through to the memory any more. IOMMU specific vulnerabilities can be grouped into four categories:

- **Boot race conditions**
  Race conditions at boot target the initial DMA Remapping (DMAR) tables in memory. This can be used to overwrite a new DMAR table, allowing certain devices to pass through the IOMMU [12]. It can also be used to misconfigure VT-d entirely, as first demonstrated by Wojtczuk et al. [60]. Markettos et al. [59] claims Apple and Microsoft and partners have patched these issues, with Microsoft using signature verification of the ACPI DMAR table during the boot process.

- **Address Translation Services (ATS)**
  PCIe provides a specification for buffering virtual address to physical address translations on the device itself. The translated address can then be used by the device directly, also setting the Address Translated (AT) flag correctly in the PCIe TLP. Previously, Linux enabled ATS by default for every device that advertised the capability to do ATS. This was later changed to only include devices that are ”trusted”, meaning only devices on internal PCIe busses. For Linux kernel versions of 4.19 and lower, ATS should still work and provide a full IOMMU bypass. This does require software device recognition, and therefore a system reboot.

- **Page Granularity**
  The IOMMU translation is page based, with a standard 4KiB granularity. This means that when the address space allocated to the PCI device is not perfectly aligned, some data gets leaked. This is not really that useful, since it is difficult to consistently and effectively utilize the leaked memory space.

- **Escalation within accessible memory**
  The accessible memory space may still be leveraged to gain code execution on the target PC. Markettos et al. [59] finds several of these attacks against several OSs. If there is kernel-level code execution, this may be utilized to disable the IOMMU, but doing so would require more extensive research and damages the integrity of the acquired memory image. Kernel Address Space Layout Randomization (KASLR) also randomizes the memory layout of the kernel, making some of these attacks even more difficult.

In the end, no known vulnerability directly gives full access to the memory on a latest version of Linux.

5.4. Proposed Improvements

Considering the limitations and vulnerabilities described in Sections 5.2 and 5.3, the flowchart in Figure 5.1 gives an overview of the steps that need to be taken to perform a succesful memory acquisition using a DMA attack using PCIe. Starting at the top of the diagram, one can follow the answers in the diagram to determine whether a DMA attack using PCIe is possible or not.

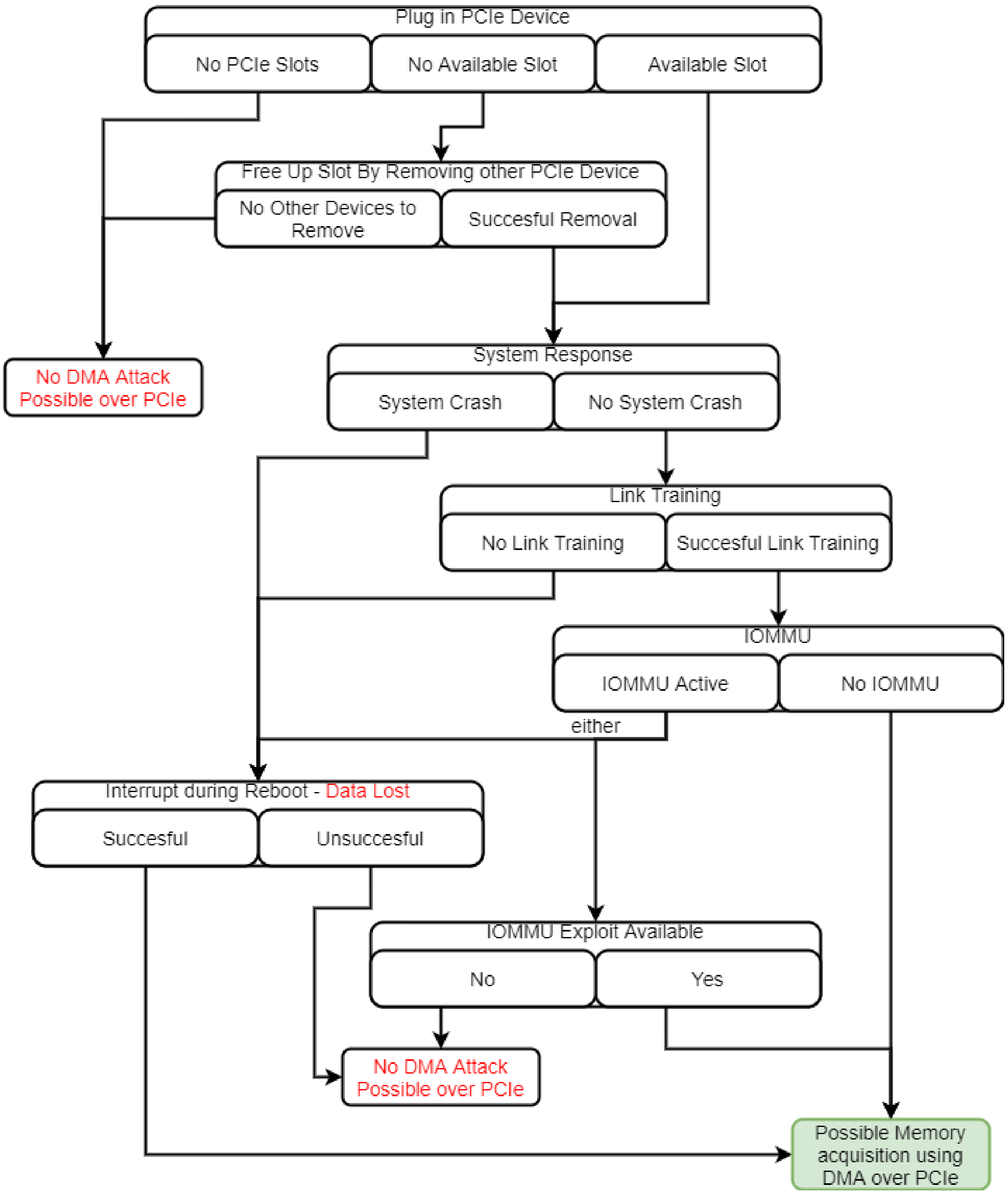


Figure 5.1: Flowchart for executing a succesful DMA Attack using PCIe for memory acquisition

5.4.1. Electrical Instability During Hot-Plug

A reboot may cause a loss of information, which is why a hot-plugged DMA attack is preferred. If the device is to be hot-plugged a number of issues arise. One of these issues is the electrical instability which may occur when a PCIe car is hot-plugged.

In order to prevent electrical instability, various solutions are proposed.

- **Externally Power Device while inserting**
  By powering the device before insertion, the card will not draw too much power from the mother-

board. This may reduce glitches in important voltage lines and prevent other devices from being affected by those glitches. A seperate extender board like the PE4H [20] can be used, or the device can be powered itself through a power-capable interface which could be present on the device.

Pin	Side B Connector		Side A Connector	
#	Name	Description	Name	Description
1	+12v	+12 volt power	PRSNT#1	Hot plug presence detect
2	+12v	+12 volt power	+12v	+12 volt power
3	+12v	+12 volt power	+12v	+12 volt power
4	GND	Ground	GND	Ground
5	SMCLK	SMBus clock	JTAG2	TCK
6	SMDAT	SMBus data	JTAG3	TDI
7	GND	Ground	JTAG4	TDO
8	+3.3v	+3.3 volt power	JTAG5	TMS
9	JTAG1	+TRST#	+3.3v	+3.3 volt power
10	3.3Vaux	3.3v volt power	+3.3v	+3.3 volt power
11	WAKE#	Link Reactivation	PERST#	PCI-Express Reset signal
Mechanical Key				
12	RSVD	Reserved	GND	Ground
13	GND	Ground	REFCLK+	Reference Clock
14	HSOp(0)	Transmitter Lane 0,	REFCLK-	Differential pair
15	HSOn(0)	Differential pair	GND	Ground
16	GND	Ground	HSIp(0)	Receiver Lane 0,
17	PRSNT#2	Hotplug detect	HSIn(0)	Differential pair
18	GND	Ground	GND	Ground

Table 5.1: Pinout for isolated PCIe extender (Adapted from pinouts.ru [52])

- **Isolation of Power Cables**
  The power to and from the device could also be completely isolated. In this case only the relevant differential pairs and the helper signals will be needed. An overview of which pins will be connected can be found in Table 5.1.

- **Measures against ESD and floating grounds**
  In order to prevent glitches in the voltage occurring through electrostatic discharge, the inserted device and the target device should have a common ground and operator inserting the device should also take precautions like wearing an ESD bracelet.

5.4.2. Enumeration During Hot-Plug

DMA requires a PCIe ID, which is obtained through the enumeration process. However, any device can perform DMA by using another PCIe ID, even if it is not its own ID. Therefore, enumeration is not needed to perform DMA. When another ID is used as the requester ID in a memory read or write TLP, it still needs to be chosen in such a way that the completion TLP is routed back to the device. Otherwise the completion with the requested data will not arrive at the device.

Bus numbers are assigned dynamically, which means that the same bus number might not refer to the same physical slot. The ID might change depending on the system configuration and which peripherals are plugged in. Brute-forcing every ID to find which ID causes a completion to be routed back to the device is possible, since the amount of possible numbers is only 32 bits. By fixing this ID to `0xffff`, the TLP is likely to be routed back through the entire PCIe network, since it is the last possible ID. IT is also very unlikely that the `0xffff` ID is used by other devices.

The PCIe device will need to be configured to pick up TLPs with the specified ID. Since the device is not enumerated, it will need to operate outside of the specification provided by PCI-SIG. The completions also contain a tag, which can be used to verify if the completion data belongs to a TLP that was sent by the device, even if another device is using that PCIe ID.

Using PCIe IDs in this way falls outside of the PCIe specification. Exact behaviour is therefore unknown and might differ per system.

5.4.3. Configuration Space

When the device is attempted to be enumerated, it will return the Vendor and Device ID. If the device is enumerated it will pollute the memory image slightly. To avoid this, the Vendor and Device ID is fixed to `0xffff`. As mentioned in Section 3.3, this will prevent enumeration.

The Vendor and Device ID can be manipulated further: they can also be changed to the Vendor and Device ID of another device. Should the device be enumerated the OS recognizes the device by the Vendor and Device ID. This means any device with a known Vendor and Device ID can be spoofed.

The configuration space also contains the PCIe capability registers. Extra capabilities like ATS can be added using these registers.

6
Validation and Results Analysis

This chapter discusses the platform used to experiment on the PCIe bus as well as perform the memory acquisition in Section 6.1. It describes the device used to execute the attack and the changes that were made to the device, based on the proposed improvements in Section 5.4

6.1. Platform Description

This section describes the hardware and software used to execute the DMA attack and the target systems it was tested on.

6.1.1. Attack Platform

The attack platform should provide the means to send TLPs on a PCIe bus, while posing as a legitimate device according to the PCIe specification. The physical device chosen in this project was a M.2 Screamer Card by LambdaConcept [27]. Using an adapter, the card can also fit in PCIe slots both 1 and 4 lanes. The gateware [31] (the HDL running on the FPGA) and accompanying software [30] called PCILeech is made by Ulf Frisk and is open source, released under a GPL 3.0 license. When talking about the gateware , this report uses ”PCILeech-FPGA”, when talking about the software, it uses ”PCILeech”.

M.2 Screamer and PCILeech-FPGA

The M.2 Screamer consists of two main parts: a USB connection to the host and the FPGA which implements the PCIe core. It can be powered partially through the JTAG interface, which is also used to program the FPGA. The open-source PCILeech-FPGA gateware allows the user fine-grained control over the FPGA and the PCIe core. The FPGA is a Xilinx 7 Series XC7A35T[72], which has a built-in PCIe core [71], which can be controlled via the FPGA through an IP provided by Xilinx. This core is not written in an HDL and cannot be altered. The core does provide information and some functionality from PCIe to the user-side on the FPGA. The PCILeech M.2 card can be seen in Figure 6.1.

There are numerous amounts of FPGAs that provide a similar functionality with regards to PCIe. The M.2 Screamer was chosen because of the already open-source project that supports it and is specifically designed for this purpose, a lower cost compared to full development kits, and the small size which allows it to be placed more easily.

The core allows changing most of the registers within the configurations space by reprogramming the core. A small portion can be changed on a live device as well. The entire configuration space can be read out to check the status of the device. The core can also be configured to let any TLP it finds through to the attacker, even TLPs with a different PCIe ID than the core.

PCILeech and LeechCore

PCIleech is open-source software build on the open-source LeechCore [29] library. LeechCore provides the interaction with the device and the tools for executing the DMA attack. PCILeech utilizes

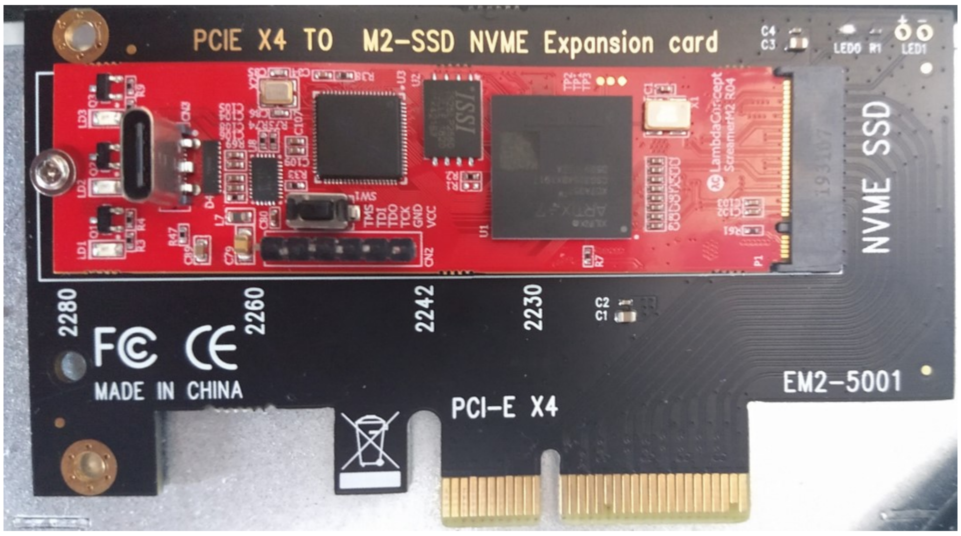


Figure 6.1: The PCILeech M.2 Card in a PCIe x4 Adapter

System	CPU	Chipset	PCIe Form Factor
Dell PowerEdge R330	Intel Xeon E3-1240	C236	x16
Dell PowerEdge R320	Intel Xeon E5-2430	C600/X79	x16
Dell Precision T5400	Intel Xeon E5410	631xESB	x8 and x16

Table 6.1: The three platforms upon which was tested

those tools to execute the attack and also contains examples for injecting modules in order to attack common OSs. It also allows to send custom TLPs on the PCIe bus, as long as the device has been completed the LTSSM properly.

6.1.2. Target Description

The acquisition method was tested on three platforms, which can be seen in Table 6.1. All platforms were running Ubuntu version `20.04.1`. The Dell Precision T5400 had two open PCIe slots. These slots displayed different behaviour.

6.2. DMA-based Memory Acquisition Using PCILeech

This section discusses the experiments that have been performed in order to validate and to attempt to improve upon DMA-based memory acquisition using PCILeech. It first discusses the electrical stability while hot-plugging the device. After the device has been successfully hot-plugged, Section 6.2.2 describes the attempt of simulating enumeration by modifying the configuration space on the device. Section 6.2.4 discusses the experiments to manipulate the PCIe ID used in the TLPs that support DMA. Section 6.2.3 takes describes manipulating the vendor and device ID in configuration space and adding PCIe capability registers. Section 6.2.5 confirms the ability to read and write to memory using TLPs. These TLPs are then used to perform a full memory acquisition, which is discussed in Section 6.2.6. The setup used in this section can be seen schematically in Figure 3.3.

6.2.1. Electrical Stability During Hot-Plug

The Electrical and LTSSM response for different set-ups was tested. The results can be seen in Table 6.2. The device was tested without modifications, with an extender cable with only the pins from Table 5.1 connected, through the PE4H Extender Board, and with the board directly powered from the JTAG interface used to program the board.

As can be seen from Table 6.2, all options decrease the chance of a system crash. However, while

		System Crash on Hot-Plug			
		Standard	**Extender Cable**	**PE4H**	**JTAG**
System	**Dell PowerEdge R330**	Sometimes, Delayed	No	No	No
	Dell PowerEdge R320	No	No	No	No
	Dell T5400 — Slot 1	Yes, Immediately	No	No	No
	Dell T5400 — Slot 2	Yes, Delayed	No	No	No
		Successful LTSSM on Hot-Plug			
		Standard	**Extender Cable**	**PE4H**	**JTAG**
	Dell PowerEdge R330	Yes	Yes	-	Yes
	Dell PowerEdge R320	No	No	No	No
	Dell T5400 — Slot 1	No	No	Yes	Yes
	Dell T5400 — Slot 2	Sometimes	Yes	Yes	Yes

Table 6.2: Results for Electrical Stability tests

the PE4H and powering the board through JTAG improve the chance at a successful link training sequence, the extender cable does this merely to a lesser extent. How exactly the upstream port detects a new device and why the extender cable does not work in this case is unknown.

6.2.2. Simulating Enumeration by Modifying the Configuration Space

When the device is hot-plugged (inserted into the system, while the system is live), the device is not enumerated. At first, an attempt was made to simulate enumeration by modifying the configuration space of the device. This could be performed by accessing the configuration space through the host PC, which controls the device on the target PC. This was unsuccessful however.

First, the enumeration sequence as performed by the Linux kernel has been analyzed. This has been performed by altering the kernel and placing extra debug information in `/drivers/pci/access.c`. The result of this extra debug information can be found in Appendix A. The PCIe Core by Xilinx provides a way to read and write to the configuration space. Using this functionality, it is attempted to perform the same actions as executed by the kernel, but now executed from the host PC.

Writing to the configuration space proved less effective. Although a part of the configuration space could be written to, most of the registers remained unchanged. In Listing 6.1, the configuration space of the device is displayed. One can note the first register in the configuration space, which is the Vendor and Device ID. In this case the Vendor ID is `0x10ee`, corresponding to Xilinx. The Device ID is `0x0666`, which corresponds to an Ethernet Adapter. By attempting to write `0xbbbb` to every register, it may become clear to what extent the configuration space can be altered live. In Appendix B, the result after attempting to write `0xbbbb` can also be seen. The IP Core by Xilinx apparently does not allow modifying every register for an unknown reason.

This method of simulating enumeration also proved flawed since the PCIe ID is not stored in the configuration space. The register containing the PCIe ID is instead only set when receiving a configuration read or write request TLP on the PCIe, upon which it assumes the ID used in that TLP. This is impossible to do through the host PC. This, combined with the fact that the configuration space limits writing to certain registers, makes simulating enumeration through the host impossible for this platform.

6.2.3. Configuration Space Manipulation

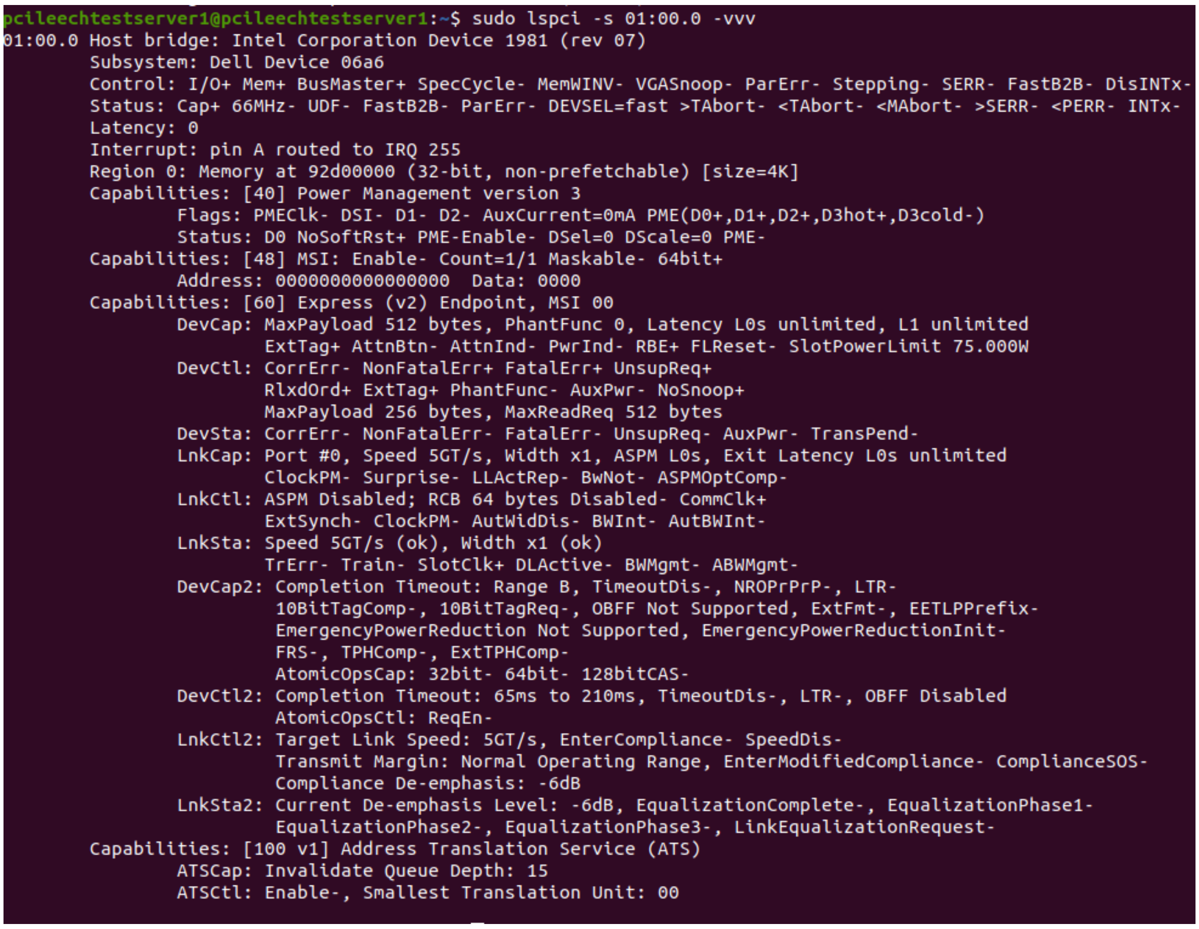
The IP Core by Xilinx has been changed to use `0xffff` as vendor and device ID. This resulted in the device not being enumerated when hot-plugged, even when a PCIe bus rescan was performed as indicated in Section 3.3. When the device is inserted before boot, the BIOS may still recognize the existence of the device, and indicates a (sometimes fatal) error with the PCIe device. For an unknown reason, this may result in not being able to boot with the device plugged in.

Listing 6.1: The PCILeech-FPGA Configuration Space before Enumeration

```
offset  data in little-endian
0000    ee 10 66 06 07 00 10 20  02 00 00 02 00 00 00 00
0010    00 00 d0 92 00 00 00 00  00 00 00 00 00 00 00 00
0020    00 00 00 00 00 00 00 00  00 00 00 00 ee 10 07 00
0030    00 00 00 00 40 00 00 00  00 00 00 00 ff 01 00 00
0040    01 48 03 78 08 00 00 00  05 60 80 00 00 00 00 00
0050    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060    10 00 02 00 e2 8f 2c 01  3e 29 00 00 12 f4 03 00
0070    40 00 12 10 00 00 00 00  00 00 00 00 00 00 00 00
0080    00 00 00 00 02 00 00 00  06 00 00 00 00 00 00 00
0090    02 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00f0    00 00 00 00 00 00 00 00  00 00 00 00 03 00 c1 10
0100    00 00 00 00 35 0a 00 01  01 00 00 00 00 00 00 00
0110    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0120    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0130    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0140    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0150    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0170    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0180    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0190    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01f0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

In a second experiment, the vendor and device ID were changed to match a Intel Host Bridge. The capability registers on the device were changed in such a way to indicate the support for ATS (Address Translation Services, mentioned in Section 3.3.2). The result can be seen in Figure 6.2. The PCIe device can still be enumerated after a rescan and is recognized as an Intel Host Bridge, as seen at the top of the Figure. The extra ATS functionality is recognized as well and can be seen at the bottom of the Fgiure. Although enabling ATS did not bypass the IOMMU on this version of Linux, it proves that this strange combination of vendor and device ID and functionality is accepted by the software nonetheless. This greatly opens up opportunities for finding bad/outdated drivers or code corresponding to the IDs and functionality.

Altering the configuration space to show a different device with different capabilities opens up possibilities for exploits within software. However, the device needs to be enumerated before these changes have actual effect. Changing the vendor and device ID to `0xffff` might also prove useful in order to hide the device from the OS. It does not mean the device is fully invisible: requesting another register from the configuration space will give a ”normal” response.



```
pcileechtestserver1@pcileechtestserver1:~$ sudo lspci -s 01:00.0 -vvv
01:00.0 Host bridge: Intel Corporation Device 1981 (rev 07)
        Subsystem: Dell Device 06a6
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0
        Interrupt: pin A routed to IRQ 255
        Region 0: Memory at 92d00000 (32-bit, non-prefetchable) [size=4K]
        Capabilities: [40] Power Management version 3
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0+,D1+,D2+,D3hot+,D3cold-)
                Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
                Address: 0000000000000000  Data: 0000
        Capabilities: [60] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s unlimited, L1 unlimited
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset- SlotPowerLimit 75.000W
                DevCtl: CorrErr- NonFatalErr+ FatalErr+ UnsupReq+
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 256 bytes, MaxReadReq 512 bytes
                DevSta: CorrErr- NonFatalErr- FatalErr- UnsupReq- AuxPwr- TransPend-
                LnkCap: Port #0, Speed 5GT/s, Width x1, ASPM L0s, Exit Latency L0s unlimited
                        ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp-
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk+
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 5GT/s (ok), Width x1 (ok)
                        TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
                DevCap2: Completion Timeout: Range B, TimeoutDis-, NROPrPrP-, LTR-
                         10BitTagComp-, 10BitTagReq-, OBFF Not Supported, ExtFmt-, EETLPPrefix-
                         EmergencyPowerReduction Not Supported, EmergencyPowerReductionInit-
                         FRS-, TPHComp-, ExtTPHComp-
                         AtomicOpsCap: 32bit- 64bit- 128bitCAS-
                DevCtl2: Completion Timeout: 65ms to 210ms, TimeoutDis-, LTR-, OBFF Disabled
                         AtomicOpsCtl: ReqEn-
                LnkCtl2: Target Link Speed: 5GT/s, EnterCompliance- SpeedDis-
                         Transmit Margin: Normal Operating Range, EnterModifiedCompliance- ComplianceSOS-
                         Compliance De-emphasis: -6dB
                LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete-, EqualizationPhase1-
                         EqualizationPhase2-, EqualizationPhase3-, LinkEqualizationRequest-
        Capabilities: [100 v1] Address Translation Service (ATS)
                ATSCap: Invalidate Queue Depth: 15
                ATSCtl: Enable-, Smallest Translation Unit: 00
```

Figure 6.2: A Device with altered configuration space as seen from the target PC (Dell PowerEdge R330)

6.2.4. PCIe ID Manipulation

PCILeech previously checked to see if the device was properly initialized by checking the PCIe ID, which is exposed to the software by the PCIe core. The software has been altered to accept and still communicate with an unenumerated device. This gives us more flexibility to communicate with the PCILeech FPGA.

The PCIe IP Core previously only accepted TLPs if they matched the internal PCIe ID of the IP Core. The only thing it accepts before that are Configuration Read and Write Requests, since those are used to enumerate the device. The IP Core has been altered to accept any incoming TLP, regardless of the PCIe ID corresponding to the TLP. One can compare this to a ”promiscuous” mode in a Network Controller.

These two changes combined successfully allow a hot-plugged and Link-Trained device to communicate on the PCIe bus of a Dell PowerEdge R330 with some specifically crafted TLPs. It has been discovered that a memory write TLP is always routed to the root complex and executed as if it were a legitimate TLP coming from a legitimate PCIe device, even if a device belonging to the PCIe ID does not exist. The only exception is when the sender is puts its PCIe ID as `0x0000`, i.e. the root complex’s PCIe ID. The TLP is rejected then.

Completions are routed back to the correct device based on a PCIe ID. Since the device now accepts any TLP presented on its bus, the only thing required to see the completion is to have it routed to the bus on which the device is present. The exact behaviour for routing TLPs with unknown PCIe IDs is unknown and most likely device-specific as well, since the PCIe specification provides no guidelines for them. It is theorized that by putting the PCIe ID as `0xffff`, the TLP will be routed furthest away and will pass through all known busses in order to find the correct device with that PCIe ID.

6.2.5. Reading and Writing Memory

By Hot-Plugging the device, waiting for Link Training, and using the adaptations to PCILeech presented in the previous section, the device can successfully perform memory read and writes. An example of a memory read can be seen in Listing 6.2. This shows a Memory Read TLP, requesting some data. The second TLP is the completion with the data attached to it.

Listing 6.2: A Memory Read TLP with its according Completion with the memory data

```
TX: MRd64:  Len: 020 ReqID: ffff BE_FL: ff Tag: 81 Addr: 0000000180000080
0000    20 00 00 20 ff ff 81 ff  00 00 00 01 80 00 00 80

RX: CplD:   Len: 020 ReqID: ffff CplID: 0000 Status: 0 BC: 080 Tag: 81
    LowAddr: 00
0000    4a 00 00 20 00 00 00 80  ff ff 81 00 2e 6c 00 00
0010    9c 3c 00 00 95 60 00 00  d2 73 00 00 d5 3f 00 00
0020    0a 01 00 00 5d 6a 00 00  7e 11 00 00 fd 74 00 00
0030    50 79 00 00 93 73 00 00  06 74 00 00 2d 56 00 00
0040    a5 31 00 00 23 54 00 00  ef 5d 00 00 e8 7c 00 00
0050    54 7a 00 00 b6 04 00 00  8f 68 00 00 3a 53 00 00
0060    07 33 00 00 e3 4c 00 00  72 49 00 00 90 53 00 00
0070    46 5b 00 00 a3 21 00 00  fd 06 00 00 cd 7a 00 00
0080    3b 0c 00 00 06 48 00 00  51 60 00 00
```

The following extra limitations were found:

- Only a maximum of 256 bytes can be requested per TLP. In a normal situation, one TLP can return an entire page (4kB) of data. Why this is not possible using an unenumerated device is not known. It also indicates that the root complex is able to differentiate between an enumerated device and an unenumerated device.
- Completions are sometimes randomly split up into multiple completions. An example can be seen in Listing 6.3. This displays two separate completions responding to the same memory request. This may be due to the link running out of credits on the Data Link Layer, or a sudden decrease in bandwidth in an upstream link.
- Certain parts of the memory cannot be read. They return a completion with a status of `1`. This can be seen in Listing 6.4, where the reponse to the memory read is stil a completion, but instead of the data, it contains a ”status” of 1, which indicates a failed completion. This may occur when reading to stolen memory space, or IOMMU protected memory space.

The kernel symbol `jiffies` was chosen to demonstrate the reading and writing capabilities of a DMA using PCIe. `jiffies` is a counter which increments based on a programmable interrupt timer. It is used to measure time since boot, and check whether the CPU is still performing normally. `jiffies` has been chosen because of the following reasons:

- It is a simple and universal value in Linux distributions which allows for verifiability, even if Linux changes.
- The value of `jiffies` changes regularly, which allows a demonstration of repeated reading of the same memory.
- The OS depends on this value and tinkering with the value is most likely going to cause unexpected system behaviour.

Listing 6.3: A Memory Read TLP with a split completion with the memory data

```
TX: MRd64:  Len: 020 ReqID: ffff BE_FL: ff Tag: 80 Addr: 0000000180000000
0000     20 00 00 20 ff ff 80 ff  00 00 00 01 80 00 00 00

RX: CplD:   Len: 010 ReqID: ffff CplID: 0000 Status: 0 BC: 080 Tag: 80
    LowAddr: 00
0000     4a 00 00 10 00 00 00 80  ff ff 80 00 0a 64 00 00
0010     e9 66 00 00 08 12 00 00  65 1c 00 00 52 25 00 00
0020     ec 70 00 00 ac 7f 00 00  b7 50 00 00 df 2f 00 00
0030     fc 7c 00 00 9b 71 00 00  cb 6c 00 00 45 69 00 00
0040     26 7e 00 00 41 63 00 00  aa 40 00 00

RX: CplD:   Len: 010 ReqID: ffff CplID: 0000 Status: 0 BC: 040 Tag: 80
    LowAddr: 40
0000     4a 00 00 10 00 00 00 40  ff ff 80 40 75 5a 00 00
0010     6e 2e 00 00 24 68 00 00  11 0d 00 00 4a 0f 00 00
0020     11 40 00 00 1a 28 00 00  d2 5e 00 00 5e 14 00 00
0030     0d 34 00 00 ba 6b 00 00  78 39 00 00 22 07 00 00
0040     e3 79 00 00 10 4d 00 00  2a 55 00 00
```

Listing 6.4: A Memory Read TLP with a failed Completion without the memory data

```
TX: MRd64: Len: 020 ReqID: aaaa BE FL: ff Tag: 80 Addr: 00000001fe000000
0000     20 00 00 20 aa aa 80 ff 00 00 00 01 fe 00 00 00

RX: Cpl: 0000   Len: 200 ReqID: aaaa CplID: 2000 Status: 1 BC: 080 Tag: 80
     LowAddr: 00
0000     0a 00 00 00 00 00 20 80 aa aa 80 00
```

Reading (and writing near) to `jiffies` using a hot-plugged device was successful. Finding the location of `jiffies` can be performed using `/proc/iomem` to locate the kernel and `boot/System.map` to locate the exact place of jiffies within the kernel.

In order verify that Jiffies works correctly, it is first read from the target PC, which can be seen in Listing 6.5. This listing displays a repeated read to the address of jiffies. It can be seen in the response that jiffies is slowly counting up. Jiffies is then read from the host PC. The result can be seen in Figures 6.3 and 6.4. The difference between the two figures is indicated in red, and it can be seen that there are two counters for jiffies, both counting up slowly.

Reading `jiffies` using PCILeech:

6.2.6. Memory Acquisition Using PCILeech

First, a partial memory range of 16MB on the target PC (The Dell PowerEdge R330) was written to using PCILeech and then read back using PCILeech and on the target server using fmem [49]. The three dumps were compared using an MD5 Hash. This way it is possible to verify the correctness of the acquisition method.

At first a file, filled with randomized data of 16MB is created using rdfc[13] and its signature verified in Listing 6.6. This file is then inserted into the target memory using PCILeech.the memory is then acquired using PCILeech and the result trimmed down to 16MB. The output file has the same hash as the input file. The memory is also read out on the target system, which was the Dell PowerEdge R330. This can be seen in Listing 6.7.

Acquiring a full memory image was partially successful. The results when reading memory can be seen in Listing 6.8. A number of pages failed to read. These pages are part of the Stolen Memory Re-

Listing 6.5: Reading Jiffies from the Target PC

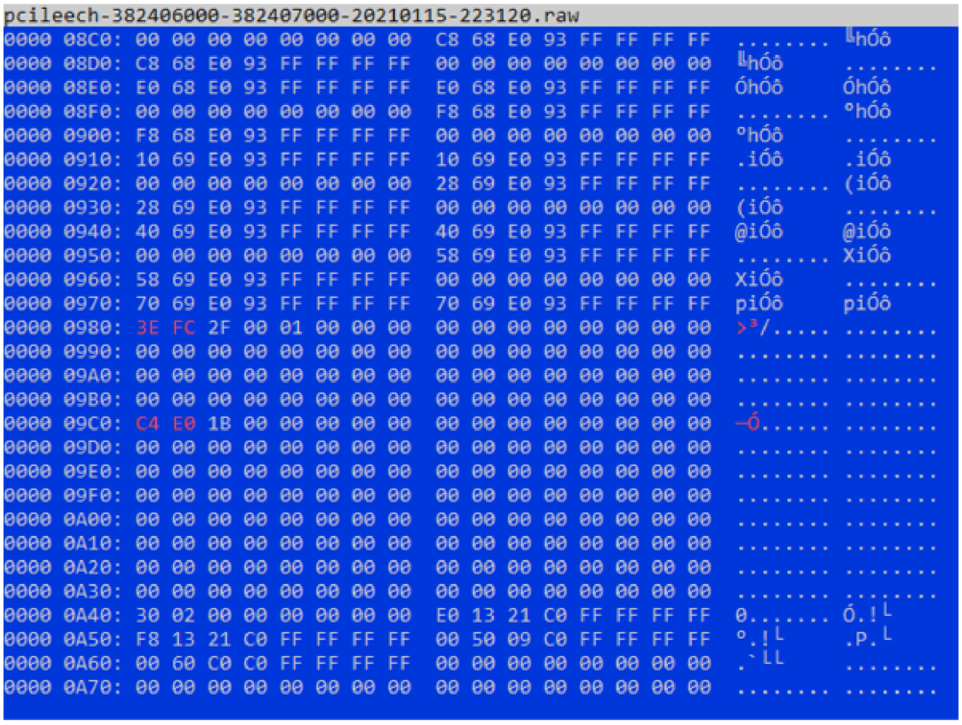
```
server1@server1:~$ sudo dd if=/dev/fmem bs=1 count=16 skip=$((0x382406980)
    ) of=temp.dump; hexdump temp.dump
16+0 records in
16+0 records out
16 bytes copied, 0,000175502 S, 91,2 kB/s
0000000 ac5a 002f 0001 0000 0000 0000 0000 0000
0000010
server1@server1:~$ sudo dd if=/dev/fnen bs=1 count=16 skip=$((0x382406980)
    ) of=temp.dump; hexdump temp.dump
16+0 records in
16+0 records out
16 bytes copied, 0,000165958 5, 96,4 kB/s
0000000 af4b 002f 0001 0000 0000 0000 0000 0000
0000010
server1@server1: $ sudo dd if=/dev/fmem bs=1 count=16 skip=$((0x382406980)
    ) of=temp.dump; hexdump temp.dump
16+0 records in
16+0 records out
16 bytes copied, 0,000196499 5, 81,4 kB/s
0000000 boe2 002f 0001 0000 0000 0000 0000 0000
0000010
server1@server1: $ sudo dd if=/dev/fmem bs=1 count=16 skip=$((0x382406980)
    ) of=temp.dump; hexdump temp.dump
16+0 records in
16+0 records out
16 bytes copied, 0,000196585 s, 81,4 kB/s
0000000 b24c 002f 0001 0000 0000 0000 0000 0000
0000010
server1@server1: $ sudo dd if=/dev/fmem bs=1 count=16 skip=$((0x382406980)
    ) of=temp.dump; hexdump temp.dump
16+0 records in
16+0 records out
16 bytes copied, 0,000180344 s, 88,7 kB/s
0000000 b340 002f 0001 0000 0000 0000 0000 0000
0000010
```

gions, as explained in Section 2.4. The exact address range can be seen in Listing 6.9. When dumping memory with an IOMMU active and a Hot-Plugged device, no pages can be read from the memory.

6.3. General Evaluation and Discussion

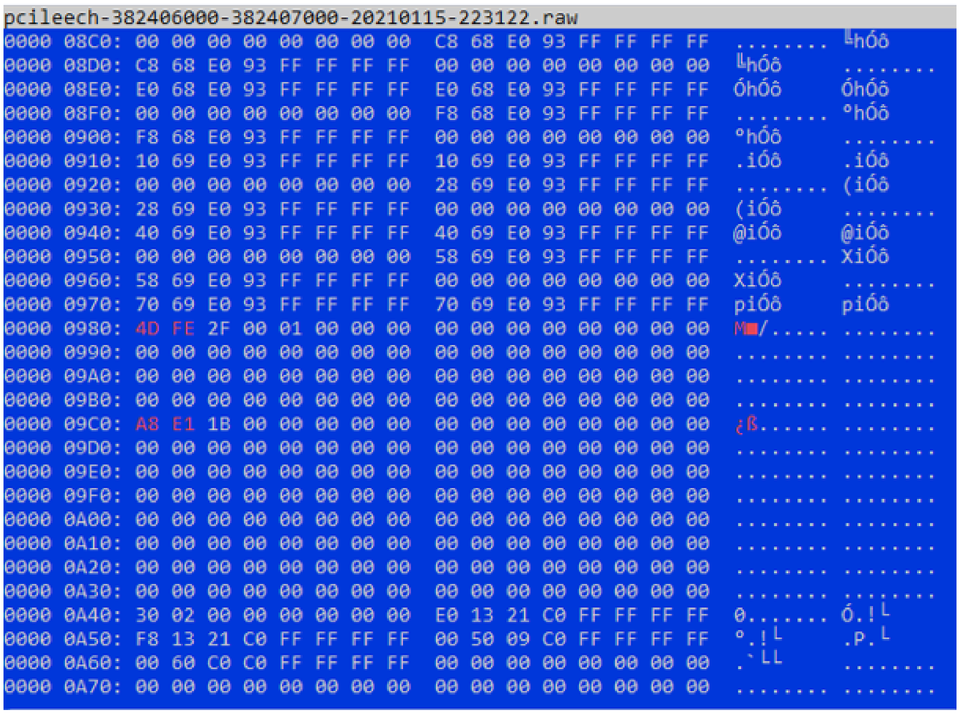
The proposed improvements have been successful in accomplishing their goal. Using a PE4H extender or powering the device through the available JTAG port prevents crashing the system. The extender cable also prevents crashing, but prevents a successful LTSSM in the Dell T5400. This is interesting since it means some systems might use other mechanisms for detecting PCIe devices than just the PRSNT pins. The Dell Poweredge R320 has a PCIe slot which does not appear to be active when it boots without a card inside the PCIe slot. The LTSSM can successfully complete when a PCIe device has been inserted during boot, but removed to free the slot for the PCILeech-fpga device.

When the LTSSM was successful, a memory acquisition could be performed. On one slot on the Dell T5400, the machine responded to memory read requests with a Vendor-Defined Message within the Message protocol of PCIe. The contents of this message indicate that the message is from an Intel



```
pcileech-382406000-382407000-20210115-223120.raw
0000 08C0: 00 00 00 00 00 00 00 00  C8 68 E0 93 FF FF FF FF  ........ ╚hÓô
0000 08D0: C8 68 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  ╚hÓô     ........
0000 08E0: E0 68 E0 93 FF FF FF FF  E0 68 E0 93 FF FF FF FF  ÓhÓô     ÓhÓô
0000 08F0: 00 00 00 00 00 00 00 00  F8 68 E0 93 FF FF FF FF  ........ °hÓô
0000 0900: F8 68 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  °hÓô     ........
0000 0910: 10 69 E0 93 FF FF FF FF  10 69 E0 93 FF FF FF FF  .iÓô     .iÓô
0000 0920: 00 00 00 00 00 00 00 00  28 69 E0 93 FF FF FF FF  ........ (iÓô
0000 0930: 28 69 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  (iÓô     ........
0000 0940: 40 69 E0 93 FF FF FF FF  40 69 E0 93 FF FF FF FF  @iÓô     @iÓô
0000 0950: 00 00 00 00 00 00 00 00  58 69 E0 93 FF FF FF FF  ........ XiÓô
0000 0960: 58 69 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  XiÓô     ........
0000 0970: 70 69 E0 93 FF FF FF FF  70 69 E0 93 FF FF FF FF  piÓô     piÓô
0000 0980: 3E FC 2F 00 01 00 00 00  00 00 00 00 00 00 00 00  >³/..... ........
0000 0990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09C0: C4 E0 1B 00 00 00 00 00  00 00 00 00 00 00 00 00  ─Ó...... ........
0000 09D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A40: 30 02 00 00 00 00 00 00  E0 13 21 C0 FF FF FF FF  0....... Ó.!└
0000 0A50: F8 13 21 C0 FF FF FF FF  00 50 09 C0 FF FF FF FF  °.!└     .P.└
0000 0A60: 00 60 C0 C0 FF FF FF FF  00 00 00 00 00 00 00 00  .`└└     ........
0000 0A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
```

Figure 6.3: Reading `jiffies` for the first time



```
pcileech-382406000-382407000-20210115-223122.raw
0000 08C0: 00 00 00 00 00 00 00 00  C8 68 E0 93 FF FF FF FF  ........ ╚hÓô
0000 08D0: C8 68 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  ╚hÓô     ........
0000 08E0: E0 68 E0 93 FF FF FF FF  E0 68 E0 93 FF FF FF FF  ÓhÓô     ÓhÓô
0000 08F0: 00 00 00 00 00 00 00 00  F8 68 E0 93 FF FF FF FF  ........ °hÓô
0000 0900: F8 68 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  °hÓô     ........
0000 0910: 10 69 E0 93 FF FF FF FF  10 69 E0 93 FF FF FF FF  .iÓô     .iÓô
0000 0920: 00 00 00 00 00 00 00 00  28 69 E0 93 FF FF FF FF  ........ (iÓô
0000 0930: 28 69 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  (iÓô     ........
0000 0940: 40 69 E0 93 FF FF FF FF  40 69 E0 93 FF FF FF FF  @iÓô     @iÓô
0000 0950: 00 00 00 00 00 00 00 00  58 69 E0 93 FF FF FF FF  ........ XiÓô
0000 0960: 58 69 E0 93 FF FF FF FF  00 00 00 00 00 00 00 00  XiÓô     ........
0000 0970: 70 69 E0 93 FF FF FF FF  70 69 E0 93 FF FF FF FF  piÓô     piÓô
0000 0980: 4D FE 2F 00 01 00 00 00  00 00 00 00 00 00 00 00  M■/..... ........
0000 0990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09C0: A8 E1 1B 00 00 00 00 00  00 00 00 00 00 00 00 00  ¿ß...... ........
0000 09D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 09F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0000 0A40: 30 02 00 00 00 00 00 00  E0 13 21 C0 FF FF FF FF  0....... Ó.!└
0000 0A50: F8 13 21 C0 FF FF FF FF  00 50 09 C0 FF FF FF FF  °.!└     .P.└
0000 0A60: 00 60 C0 C0 FF FF FF FF  00 00 00 00 00 00 00 00  .`└└     ........
0000 0A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
```

Figure 6.4: Reading `jiffies` for a second time

device. The message does not contain the memory data and the exact meaning of the message is unknown. To what extent the improvements were successful can be seen in Table 6.3.

These experiments have only been tested on three different systems and more testing on a wider variety of systems is necessary to draw a stronger conclusion on the viability of a Hot-Plugged DMA-based memory acquisition. Although improvements were made on these systems, the process showed

Pin	Side B Connector		Side A Connector	
#	Name	Description	Name	Description
1	+12v	+12 volt power	PRSNT#1	Hot plug presence detect
2	+12v	+12 volt power	+12v	+12 volt power
3	+12v	+12 volt power	+12v	+12 volt power
4	GND	Ground	GND	Ground
5	SMCLK	SMBus clock	JTAG2	TCK
6	SMDAT	SMBus data	JTAG3	TDI
7	GND	Ground	JTAG4	TDO
8	+3.3v	+3.3 volt power	JTAG5	TMS
9	JTAG1	+TRST#	+3.3v	+3.3 volt power
10	3.3Vaux	3.3v volt power	+3.3v	+3.3 volt power
11	WAKE#	Link Reactivation	PERST#	PCI-Express Reset signal
	Mechanical Key			
12	RSVD	Reserved	GND	Ground
13	GND	Ground	REFCLK+	Reference Clock
14	HSOp(0)	Transmitter Lane 0,	REFCLK-	Differential pair
15	HSOn(0)	Differential pair	GND	Ground
16	GND	Ground	HSIp(0)	Receiver Lane 0,
17	PRSNT#2	Hotplug detect	HSIn(0)	Differential pair
18	GND	Ground	GND	Ground

Table 6.3: Improvements to memory acquisition on various systems

a lot of quirks. Instances like the Extender Cable preventing the LTSSM indicate the need for more research into how the chipset actually functions. Most of the techniques here are not within the PCIe specification. The specification even explicitly forbids a device from using a Vendor and Device ID of `0xffff`. This means it is difficult to say whether these techniques actually hold up outside of the currently tested systems.

The memory dumps that can be made are also verified to be true to the memory image using the MD5 Hash. However, this only concerns memory the memory acquisition method can reach. Using `/proc/iomem`, which can be seen in Appendix C one can confirm the memory mapping of the main memory. PCILeech indicates a failure in reading pages from `0x8f000000` to `0x100000000`. `/proc/iomem` shows that this memory range contains the following data:

- Memory reservations for the software for PCIe devices
- Memory reservations for the configuration spaces of PCIe devices
- Memory related to the interrupt controller for IO devices
- Memory for the HPET (High Precision Event Timer)
- Memory for plug-and-play devices like USB devices

The kernel is still located outside of this area, which means it is still possible to access it like proven in Section 6.2.5.

As proven in Section 6.2.5 with reading `jiffies`, this memory acquisition method is not perfectly atomic. The acquisition speed is roughly 26MB/s. Memory acquisitions using an enumerated device are faster, because in that case the maximum amount of memory data that is being received per MRd TLP is 4KiB (one page). An unenumerated method is therefore less atomic. An unenumerated method is better in regards of correctness. An enumerated device takes up a small amount memory on the target itself through MMIO, which does not happen with an unenumerated device. The integrity of the method is verified using the experiment in Section 6.2.6. Data can be acquired from the memory and is the same from both the point of view from the target, as well as the host system.

Listing 6.6: Validating a memory dump from the Host PC

```
The Hash of the randomly generated file:
----------------------------------------

pcileech\files>certutil -hashfile corr.test MD5
MD5 hash of corr.test:
520f676e713c4ed64e48ffcda542638f
CertUtil: -hashfile command completed successfully.

The file is inserted into the target memory:
--------------------------------------------

pcileech\files>pcileech.exe write -min 0x180000000 -in 16mb2.test

Memory Write: Successful.

pcileech\files>pcileech.exe dump -min 0x180000000 -max 0x190000000

Current Action: Dumping Memory
Access Mode: Normal
Progress: 256 / 256 (100%)
Speed: 85 MB/s
Address: 0x0000000190000000
Pages read: 65536 / 65536 (100%)
Pages failed: 0 (0%)
Memory Dump: Successful.

The memory dump is trimmed down to 16MB
---------------------------------------

pcileech\files >dd if=pcileech - 180000000-190000000-20210320-220748.raw
   of=corr5.test bs=1 count=16000000
rawwrite dd for windows version 0.6beta3.
Written by John Newbigin <jn@it.swim.edu.au>
This program is covered by terms of the GPL Version 2.

16000000+0 records in
16000000+0 records out

The output file is hashed again and shows the correct Hash
----------------------------------------------------------

pcileech\files>certutil -hashfile corr5.test MD5
MD5 hash of corr5.test:
520f676e713c4ed64e48ffcda542638f
CertUtil: -hashfile command completed successfully.
```

Listing 6.7: Validating a memory dump from the Target PC

```
Retrieving the data from memory using fmem on the target PC:
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

server1@server1:~/Documents/fmem/fmem$ dd if=/dev/fmem of=temp.dump bs=1
   count=16000000 skip=$((0x180000000)); hexdump temp.dump −n40
16000000+0 records in
16000000+0 records out
16000000 bytes (16 MB, 15 MiB) copied, 32,3772 s, 494 kB/s
0000000 640a 0000 66e9 0000 1208 0000 1c65 0000
0000010 2552 0000 70ec 0000 7fac 0000 50b7 0000
0000020 2fdf 0000 7cfc 0000
0000028

Computing the hash of the acquired memory using md5sum:
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

server1@server1:~/Documents/fmem/fmem$ md5sum temp.dump
520f676e713c4ed64e48ffcda542638f temp.dump
```

Listing 6.8: Reading full memory from target

```
>pcileech.exe probe
Initial DeviceID check Failed
Unable to retrieve required Device PCIe ID, Spoof ID in software? (y/n)y

Memory Map:
START                          END                                  #
   PAGES
0000000000000000        000000000009ffff        0000000
000000000000000         000000008effffff        0008ef40
0000000100000000        000000046fffffff        00370000
Current Action: Probing Memory
Access Mode: Normal
Progress: 18176 / 18176 (100%)
Speed:
370 MB/S
Address:
0x0000000470000000
Pages read: 4190176 / 4653056 (90%)
Pages failed:
462880 (9%)
Memory Probe: Completed.
```

Listing 6.9: Reading failing memory from target

```
>pcileech.exe probe −max 0x100000000 −min 0x8f000000
Initial DeviceID check Failed
Unable to retrieve required Device PCIe ID, Spoof ID in software? (y/n)
y
Memory Map:
START     END                 #PAGES
Current Action: Probing Memory
Access Mode: Normal
Progress: 1808 / 1808 (100%)
Speed:
361 MB/s
Address: 0x00000000000000
Pages read:
0 / 462848 (0%)
Pages failed:
462848 (100%)
Memory Probe: Completed.
```

7 Conclusion

This chapter provides a summary of the thesis in Section 7.1 and recommendations for future work in Section 7.2.

7.1. Summary

Chapter 1: Introduction describes the motivation behind the thesis, followed by the context of digital forensics to which this project developed. The state-of-the-art analysis outlines the past developments regarding memory analysis and current memory acquisition techniques. It concludes with a description of the contribution of the thesis and the general structure for the rest of the thesis.

Chapter 2: Memory Organization provides an up-to-date overview of modern memory organization, discussing the relevant concepts with regards to the three main building blocks of modern day storage: the cache, the main memory and permanent storage. It also discusses memory controllers and their impact on computer system memory, as well as the interfaces which can connect to the DRAM.

Chapter 3: Peripheral Component Interconnect Express discussed the basics of PCI express and the three layers that constitute the protocol. It describes how a PCIe device is initialized and can be configured. Afterwards, Direct Memory Access (DMA) using PCIe is discussed. The relevant TLPs and their purpose within DMA are explained.

Chapter 4: Memory Acquisition Techniques discusses the different existing techniques or concepts of techniques for acquiring memory. It classifies them into three categories based on the interface where the acquisition is executed. The first type of acquisition technique is software based. This includes running native or 3rd party software tools on the system itself to provide a memory snapshot. The second category uses DMA and provides a straightforward way of accessing the memory. This category is divided into two subcategories: if the DMA channel can be directly accessed by a device the attacker chooses, it is classified as a direct DMA channel. If it requires subverting an existing device with a DMA channel, it is classified as an indirect DMA channel. The memory can also be accessed directly over the memory bus and also can be targeted directly, which constitute the last category of memory acquisition method. In contrast to software-based memory acquisition techniques, hardware-based techniques have the advantage that they do not require admin access or a login.

In order to limit the scope of the thesis, a scenario in which a solution would need to operate is described. Next, an adapted model is used to determine to most effective method of memory acquisition. This chapter discusses the reasoning behind the model, applies it to available hardware methods, and discusses the result.

Chapter 5: DMA-based Memory Acquisition Using PCIe further details what a DMA attack exactly is. It first discusses past usages of DMA and a state-of-the-art of modern DMA attacks. It then describes the main countermeasures and drawbacks of acquiring memory based on these attacks.

Based on a vulnerability analysis of PCIe and DMA, improvements to the memory acquisition method have been proposed. These improvements intend to facilitate a hot-plug of the attack device by ensuring electrical stability of the system and a device that functions without enumeration. By changing the vendor and device ID in configuration space, it is possible to make sure that enumeration does not happen, which ensures an untouched memory image.

Chapter 6: Validation and Results Analysis discusses PCILeech: the platform used to experiment on the PCIe bus, as well as perform the memory acquisition. It described the device used to execute the acquisition and the changes that were made to the device, based on the proposed improvements in Chapter 5.

Section 6.2 discusses the experiments that have been performed in order to validate and to attempt to improve upon DMA-based memory acquisition using PCILeech. It first discusses the electrical stability while hot-plugging the device. After the device has been successfully hot-plugged, it describes the attempt of simulating enumeration by modifying the configuration space on the device. Afterwards, it describes the experimentation to manipulate the PCIe ID used in the TLPs that support DMA and to manipulate the vendor and device ID in configuration space and adding PCIe capability registers. The ability to read and write to memory using TLPs is displayed and these TLPs are then used to successfully perform a full memory acquisition.

Memory acquisition using DMA over PCIe was successful. The proposed improvements provided a way to perform DMA without having to reboot the target system and have less impact on it. These improvements can aid a forensic investigator in performing a better memory analysis.

7.2. Future Work

This thesis proposes several areas of potential future work.

- **LTSSM / PCH / Switches / Chipset interaction**
  What exactly triggers Link Training after a Hot-Plug and how the interaction is handled between the downstream device and an upstream device is still unknown. It would be insightful to further research how the device interacts with several upstream devices like the PCH, a PCIe switch or the chipset.
- **IOMMU**
  When the IOMMU is active and correctly implemented, it can counter a DMA attack almost completely. Further research into the IOMMU and its interaction with PCIe may prove insightful to bypass (some parts of) the IOMMU.
- **DMI**
  The DMI bus connects the PCH to the CPU and functions as an extension of the PCIe network. the DMI bus is PCIe bus 0. This means all packets pass through this bus. It also connects directly to the Root Complex and the IOMMU also communicates to the Root Complex over this bus. Further understanding of this bus and the possibility to connect to it might lead to finding interesting exploits.
- **Warm reboot**
  A reboot might destroy (parts of) the memory. A ”warm” reboot keeps the RAM and CPU turned on. This may keep the memory data intact after a reboot. Further research into the effects of a reboot on the RAM and possible system subversion during a system reboot can lead to a better acquisition technique.
- **SMBus / JTAG**
  The PCIe interface automatically includes a JTAG and SMBus interface. It is currently not know exactly what impact these protocols could have and to what extent they can influence upstream devices.

- **Cache coherency**
  Reading and Writing to RAM may be successful, but there might also be data in the CPU. The cache within the CPU is usually not fully up-to-date with the main memory. If the RAM is changed after a memory write through DMA, will the cache be updated automatically? The same could be true for the reverse: Is there newer memory data within the cache, which is currently not the same in the main memory. Further research may reveal information about the coherency between the cache and the CPU during a DMA-based acquisition.

- **Different systems**
  The memory acquisition has been shown to work on these specific systems, but testing the method on a higher amount of systems, with other CPUs and/or chipsets will yield a stronger conclusion about the robustness, flexibility, stability, and applicability of the method.

- **Different PCIe slots**
  The device has currently only been inserted in any available PCIe slot. However, some slots are connected to a switch, others to a PCH, and others directly to the root complex. Investigating the behaviour of the device in different slots can lead to a better understanding of the acquisition method.

- **PCIe device detection**
  The experiments in Section 6.2.1 showed a difference in device detection when the PE4H connector was used compared to the extender cable. This behaviour also differed per tested system. Finding out how a system detects and sets up a hot-plugged PCIe device can lead to a more stable hot-plug method.

Appendices

A
Enumeration Kernel Debug

A short excerpt with explanation of some dumped debug information during a re-enumeration sequence in the Linux kernel. This debug information was acquired by changing `/drivers/pci/access.c` and recompiling the Linux Kernel. This new kernel was run on the Dell PowerEdge R330.

A print statement has been added in PCI_OP_READ and PCI_OP_WRITE, which are the base functions any driver uses to read and write to pci devics. The information in the debug print statement is as follows:

- Function: Where the debug statement is in the code
- Devfn: The device function number that is being addressed, part of the full PCIe ID
- Pos: The position of the register that is being accessed
- Len: The length of the register(s) that is being accessed in bytes
- Data: The data that is read from the device
- &data: where the data is stored on the target system
- Value: The data written to the device

```
[  102.712778] Function: PCI_OP_READ, devfn is 11, pos is 0, len is 4,
    data is ffffffff ,
Explanation:
Target: The device ID and Vendor ID register
What it does: Attempt to read out some device on some bus. If the result
    is ffffffff , this indicates that the device does not exist (or has
    ffffffff as a device/vendor ID)

[  102.729858] Function: PCI_OP_READ, devfn is 0, pos is 0, len is 4, data
     is 66610ee
Explanation:
Target: The device ID and Vendor ID register
What it does: Attempt to read out some device on some bus. In this case,
    the device and vendor ID have been returned. The Vendor ID is 10ee,
    corresponding to Xilinx Corp. The Device ID is 0666 (Ethernet Adapter)

 [  102.729941] pci 0000:01:00.0: Function: pci_write_config_word
[  102.729945] Function: PCI_OP_WRITE, devfn is 0, pos is 4, len is 2,
    value is 400
Explanation:
Target: Command Register
```

```
What it does: Enable Fast Back-to-Back transfers

 [   102.730191] pci 0000:01:00.0: Function: pci_write_config_word
[   102.730195] Function: PCI_OP_WRITE, devfn is 0, pos is 104, len is 2,
    value is 2930
Explanation:
Target: Device Control Register in extended capabilities
What it does: Change max payload size from 128 to 256
```

B Writing to Configuration Space Using the IP Core

The configuration space before enumeration can be seen in Listing B.1. Next, in order to demonstrate the extent to which this configuration space can be written to, it is attempted to write 0xbbbb to every data word in the configuration space. The result can be seen in Listing B. It can be seen that barely any registers are affected by this write.

Listing B.1: The PCILeech-FPGA Configuration Space before Enumeration

```
offset  data in little-endian
0000    ee 10 66 06 07 00 10 20  02 00 00 02 00 00 00 00
0010    00 00 d0 92 00 00 00 00  00 00 00 00 00 00 00 00
0020    00 00 00 00 00 00 00 00  00 00 00 00 ee 10 07 00
0030    00 00 00 00 40 00 00 00  00 00 00 00 ff 01 00 00
0040    01 48 03 78 08 00 00 00  05 60 80 00 00 00 00 00
0050    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060    10 00 02 00 e2 8f 2c 01  3e 29 00 00 12 f4 03 00
0070    40 00 12 10 00 00 00 00  00 00 00 00 00 00 00 00
0080    00 00 00 00 02 00 00 00  06 00 00 00 00 00 00 00
0090    02 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00f0    00 00 00 00 00 00 00 00  00 00 00 00 03 00 c1 10
0100    00 00 00 00 35 0a 00 01  01 00 00 00 00 00 00 00
0110    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0120    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0130    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0140    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0150    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0170    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0180    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0190    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01f0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
01e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
offset  data in little-endian
0000    ee 10 66 06 03 01 10 00  02 00 00 02 bb 00 00 00
0010    00 b0 bb bb 00 00 00 00  00 00 00 00 00 00 00 00
0020    00 00 00 00 00 00 00 00  00 00 00 00 ee 10 07 00
0030    00 00 00 00 40 00 00 00  00 00 00 00 bb 01 00 00
0040    01 48 03 78 0b 1b 00 00  05 60 b1 00 b8 bb bb bb
0050    bb bb bb bb bb bb 00 00  00 00 00 00 00 00 00 00
0060    10 00 02 00 e2 8f b8 0b  bb 3b 11 00 12 f4 03 00
0070    9b 0b 12 10 00 00 00 00  bb 13 a0 00 1b 00 00 00
0080    00 00 00 00 02 00 00 00  1b 03 00 00 00 00 00 00
0090    bb 1b 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00f0    00 00 00 00 00 00 00 00  00 00 00 00 03 00 c1 10
0100    00 00 00 00 35 0a 00 01  01 00 00 00 00 00 00 00
0110    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0120    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0130    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0140    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0150    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0170    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0180    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0190    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01a0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01b0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01c0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01d0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01e0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01f0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

C
Memory Mapping with `/proc/iomem`

Using `/proc/iomem` one can confirm the memory mapping of the main memory. PCILeech indicates a failure in reading pages from `0x8f000000` to `0x100000000`. Through `/proc/iomem` we see that this memory mainly contains MMIO related memory contents. The kernel is still located outside of this area, which means we can still access it like proven in Section 6.2.5.

Listing C.1: The content of `/proc/iomem` for a memory dump

```
00000000-00000fff : Reserved
00001000-00057fff : System RAM
00058000-00058fff : Reserved
00059000-0009ffff : System RAM
000a0000-000bffff : PCI Bus 0000:00
000f0000-000fffff : System ROM
00100000-857ae017 : System RAM
857ae018-857de057 : System RAM
857de058-857df017 : System RAM
857df018-8580f057 : System RAM
8580f058-85c09017 : System RAM
85c09018-85c21457 : System RAM
85c21458-85d0f017 : System RAM
85d0f018-85d17057 : System RAM
85d17058-8d480fff : System RAM
8d481000-8d481fff : ACPI Non-volatile Storage
8d482000-8d4cbfff : Reserved
8d4cc000-8d613fff : System RAM
8d614000-8ef69fff : Reserved
  8d6c4b98-8d6c4b98 : wdat_wdt
  8e303018-8e303067 : APEI ERST
  8e303070-8e303077 : APEI ERST
  8e303078-8e305017 : APEI ERST
8ef6a000-8ef99fff : ACPI Non-volatile Storage
8ef9a000-8efe6fff : ACPI Tables
8efe7000-8effffff : System RAM
8f000000-8fffffff : RAM buffer
90000000-dfffffff : PCI Bus 0000:00
  90000000-9001ffff : pnp 00:08
  90100000-901fffff : PCI Bus 0000:04
    90100000-9013ffff : 0000:04:00.0
    90140000-9017ffff : 0000:04:00.1
  91000000-91ffffff : PCI Bus 0000:05
    91000000-91ffffff : PCI Bus 0000:06
```

```
      91000000-91ffffff : PCI Bus 0000:07
        91000000-91ffffff : PCI Bus 0000:08
          91000000-91ffffff : 0000:08:00.0
          91000000-91ffffff : mgadrmfb_vram
  92000000-929fffff : PCI Bus 0000:05
    92000000-929fffff : PCI Bus 0000:06
      92000000-928fffff : PCI Bus 0000:07
        92000000-928fffff : PCI Bus 0000:08
          92000000-927fffff : 0000:08:00.0
          92800000-92803fff : 0000:08:00.0
          92800000-92803fff : mgadrmfb_mmio
  92a00000-92afffff : PCI Bus 0000:04
    92a00000-92a0ffff : 0000:04:00.1
      92a00000-92a0ffff : tg3
    92a10000-92a1ffff : 0000:04:00.1
      92a10000-92a1ffff : tg3
    92a20000-92a2ffff : 0000:04:00.1
      92a20000-92a2ffff : tg3
    92a30000-92a3ffff : 0000:04:00.0
      92a30000-92a3ffff : tg3
    92a40000-92a4ffff : 0000:04:00.0
      92a40000-92a4ffff : tg3
    92a50000-92a5ffff : 0000:04:00.0
      92a50000-92a5ffff : tg3
  92b00000-92cfffff : PCI Bus 0000:03
    92b00000-92bfffff : 0000:03:00.0
    92c00000-92c0ffff : 0000:03:00.0
      92c00000-92c0ffff : megasas: LSI
    92c20000-92c3ffff : 0000:03:00.0
  92d00000-92d0ffff : 0000:00:14.0
    92d00000-92d0ffff : xhci-hcd
  92d10000-92d13fff : 0000:00:1f.2
  92d14000-92d15fff : 0000:00:17.0
    92d14000-92d15fff : ahci
  92d16000-92d167ff : 0000:00:17.0
    92d16000-92d167ff : ahci
  92d17000-92d170ff : 0000:00:17.0
    92d17000-92d170ff : ahci
e0000000-efffffff : PCI MMCONFIG 0000 [bus 00-ff]
  e00fa000-e00fafff : Reserved
  e00fd000-e00fdfff : Reserved
fd000000-fe7fffff : PCI Bus 0000:00
  fd000000-fdabffff : pnp 00:00
  fdad0000-fdadffff : pnp 00:00
  fdc00000-fdcfffff : Reserved
    fdc6000c-fdc6000c : wdat_wdt
  fe000000-fe010fff : Reserved
  fe036000-fe03bfff : pnp 00:00
  fe03d000-fe3fffff : pnp 00:00
  fe410000-fe7fffff : pnp 00:00
fec00000-fec003ff : IOAPIC 0
fed00000-fed003ff : HPET 0
  fed00000-fed003ff : PNP0103:00
fed10000-fed17fff : pnp 00:08
fed18000-fed18fff : pnp 00:08
fed19000-fed19fff : pnp 00:08
```

```
fed20000-fed3ffff : pnp 00:08
fed45000-fed8ffff : pnp 00:08
fed90000-fed93fff : pnp 00:08
fee00000-feefffff : pnp 00:08
  fee00000-fee00fff : Local APIC
ff000000-ffffffff : INT0800:00
  ff400000-ffffffff : Reserved
100000000-46fffffff : System RAM
  421400000-422402496 : Kernel code
  422600000-422ecdfff : Kernel rodata
  423000000-4233662ff : Kernel data
  423625000-423bfffff : Kernel bss
2000000000-2fffffffff : PCI Bus 0000:00
  2ffff00000-2ffff000ff : 0000:00:1f.4
  2ffff01000-2ffff01fff : 0000:00:16.1
  2ffff02000-2ffff02fff : 0000:00:16.0
  2ffff03000-2ffff03fff : 0000:00:14.2
    2ffff03000-2ffff03fff : Intel PCH thermal driver
```

D Common Server Configurations

In order to better understand the landscape of server models and configurations, an exploration has been performed to analyze the current best selling servers on Amazon [7]. If the server configuration contains the mentioned property, the value will be `1`, otherwise `0`. For example, if a server has an RDIMM form factor, the value in the table under RDIMM for that particular server is `1`. The numbering on the left-hand side of the table is the same for every table. From this analysis it is clear that Intel is currently dominant with regards to CPUs and that the only common user interfaces are PCIe and some USB interface.

	Server			
	Brand	Type	TypeNr.	Generation
1	HP	Proliant	DL360p	8
2	Dell	Poweredge	T40	
3	Dell	Poweredge	R720	
4	HP	Proliant	Microserver	10
5	HP	Proliant	DL160	6
6	HP	Proliant	DL360p	7
7	HP		Z440	
8	Dell	Poweredge	R810	
9	HP	Proliant	ML350	10
10	Dell	Poweredge	R820	
11	Dell	Poweredge	T320	
12	Dell	Poweredge	R620	
13	HP	Proliant	ML110	10
14	HP		BL460c	10
15	Dell	Poweredge	T340	
16	Dell	Poweredge	R610	
17	Dell	Poweredge	T20	
18	Dell	Poweredge	2950	
19	Lenovo	Thinkstation	P500	
20	HP	Proliant	DL360	

Table D.1: Common server types. The number on the left is the same number used in the next tables.

	CPU				RAM		
	Brand	Type	TypeNr.	Microarchitecture	RDIMM	LRDIMM	UDIMM
1	Intel	Xeon	E5-2640	Sandy Bridge	1	0	0
2	Intel	Xeon	E-2224G	Coffee Lake	0	0	1
3	Intel	Xeon	E5-2670	Sandy Bridge	1	0	0
4	Intel	Xeon	E-2224	Coffee Lake	0	0	1
5	Intel	Xeon	X5650	Westmere	1	0	1
6	Intel	Xeon	X5650	Westmere	1	0	1
7	Intel	Xeon	E5-2630	Sandy Bridge	1	0	1
8	Intel	Xeon	E7-4870	Westmere	1	0	1
9	Intel	Xeon	Gold 5118	Skylake	1	0	0
10	Intel	Xeon	E5-4650	Sandy Bridge	1	0	0
11	Intel	Xeon	E5-2430	Sandy Bridge	1	0	0
12	Intel	Xeon	E5-2660	Sandy Bridge	1	1	1
13	Intel	Xeon	Silver 4210	Cascade Lake	1	0	0
14	Intel	Xeon	Gold 5120	Skylake	1	1	0
15	Intel	Xeon	E-2124	Coffee Lake	0	0	1
16	Intel	Xeon	E5540	Nehalem	1	0	1
17	Intel	Xeon	E3-1225 v5	Skylake	0	0	1
18	Intel	Xeon	5130	Woodcrest	1	0	0
19	Intel	Xeon	E5-2620	Haswell	1	1	0
20	Intel	Xeon	Gold 6248	Cascade Lake	1	1	0

Table D.2: Common CPU configurations and RAM Form Factors

	Management		Network Controller		Storage Controller		
	Type	Firmware	Brand	Type	Brand	Type	TypeNr.
1	HP iLO	HP iLO 4	HP	331FLR	HP	Smart Array	P420i
2	None	None	Intel	I219	Intel	VROC	6.X
3	Intel iDRAC7	iDRAC 7	Intel	I350	Dell	PERC	H710P
4	HP iLO	HP iLO 5	Intel	i350 AM4	HP	Smart Array	S100i SR Gen 10
5	HP iLO	100i	HP	NC362i	HP	Smart Array	B110i
6	HP iLO	HP iLO 3	HP	NC382i	HP	Smart Array	P410i
7	None	None	Intel	I218	LSI	None	9217-4i
8	Intel iDRAC6	iDRAC6	Broadcom	NetExtreme II 5709	Dell	Perc	H700
9	HP iLO	HP iLO 5	HP	369i	HP	Smart Array	P408i-a
10	Intel iDRAC 7	iDRAC 7	Broadcom	5720	Dell	PERC	H710
11	Intel iDRAC7	iDRAC 7	Broadcom	5720	Dell	PERC	H710
12	Intel iDRAC	iDRAC 7	Broadcom	5720	Dell	PERC	H310
13	HP iLO	HP iLO 5	HP	331T	HP	Smart Array	S100i SR Gen 10
14	HP iLO	HP iLO 5	HP	FlexFabric 536FLB	HP	Smart Array	S100i SR Gen 10
15	Intel iDRAC	iDRAC 9	Broadcom	5720	Dell	PERC	S140
16	Intel iDRAC	iDRAC 6	Broadcom	5709	Dell	PERC	6i
17	Intel AMT	AMT 11.0	Intel	I219-LM	Intel	Rapid Storage	12.x
18	BMC		Broadcom	Qlogic 5708	Dell	PERC	5i
19	Intel AMT	AMT 9	Intel	i218LM	None		
20	HP iLO	iLO 5	Unknown		HP	Smart Array	S100i

Table D.3: Common management engines, network controllers and storage controllers

	HDD					PCIe							
	SATA	SAS	LFF	SFF	uFF	1.0	2.0	3.0	x16	x8	x4	PCI	PCI-X
1	0	1	0	1	0	0	0	1	1	1	0	0	0
2	1	0	1	0	0	0	0	1	1	0	1	1	0
3	1	1	1	0	0	0	0	1	1	1	0	0	0
4	1	0	1	0	0	0	0	1	1	0	0	0	0
5	1	1	1	1	0	0	1	0	1	0	1	0	0
6	1	1	0	1	0	0	1	0	1	1	0	0	1
7	1	1	0	1	0	0	1	1	1	1	1	1	0
8	1	1	0	1	0	0	1	0	1	1	1	0	0
9	1	1	0	1	0	0	0	1	1	1	1	0	0
10	1	1	0	1	0	0	0	1	1	1	0	0	0
11	1	1	1	0	0	0	0	1	1	1	0	0	0
12	1	1	0	1	0	0	0	1	1	1	0	0	0
13	1	1	1	1	0	0	0	1	1	1	1	0	0
14	1	1	0	1	0	0	0	1	1	0	0	0	0
15	1	1	1	0	0	0	0	1	1	1	0	0	0
16	1	1	0	1	0	0	1	0	0	1	1	0	0
17	1	0	0	0	0	0	0	1	1	0	1	1	0
18	1	1	0	1	0	0	1	0	0	1	0	1	0
19	1	1	1	0	0	0	1	1	1	0	1	1	0
20	1	1	0	0	1	0	0	1	1	1	0	0	0

Table D.4: Common server configurations for the storage and PCIe interfaces

	On-Board Interfaces			User-Interfaces (non-USB)			
	Intel QPI	Intel UPI	FSB	RJ45	PS2	Thunderbolt	IEEE 1394 (FireWire)
1	1	0	0	1	0	0	0
2	0	1	0	1	1	0	0
3	1	0	0	1	0	0	0
4	0	1	0	1	0	0	0
5	1	0	0	1	0	0	0
6	1	0	0	1	0	0	0
7	1	0	0	1	1	0	0
8	1	0	0	1	0	0	0
9	0	1	0	1	0	0	0
10	1	0	0	1	0	0	0
11	1	0	0	1	0	0	0
12	1	0	0	1	0	0	0
13	0	1	0	1	0	0	0
14	0	1	0	1	0	0	0
15	0	1	0	1	0	0	0
16	1	0	0	1	0	0	0
17	0	1	0	1	0	0	0
18	0	0	1	1	0	0	0
19	1	0	0	1	1	1	1
20	0	1	0	1	0	0	0

Table D.5: Common server configurations for the on-board and non-USB user interfaces

	User Interfaces - USB					
	USB-B	USB-C	USB 2.0	USB 3.0	USB 3.1	USB 3.2
1	0	0	1	0	0	0
2	0	1	1	1	1	0
3	0	0	1	0	0	0
4	0	0	1	0	0	1
5	0	0	1	0	0	0
6	0	0	1	0	0	0
7	0	0	1	1	0	0
8	0	0	1	0	0	0
9	0	0	1	1	0	0
10	0	0	1	0	0	0
11	0	0	1	0	0	0
12	0	0	1	0	0	0
13	0	0	1	1	0	0
14	0	0	1	0	0	0
15	0	0	1	1	0	0
16	0	0	1	0	0	0
17	0	0	1	1	0	0
18	0	0	1	0	0	0
19	0	0	1	1	0	0
20	0	0	1	1	0	0

Table D.6: Common server configurations for USB interfaces

Bibliography

[1] The authoritative dictionary of ieee standards terms, seventh edition. *IEEE Std 100-2000*, pages 1–1362, 2000. doi: 10.1109/IEEESTD.2000.322230.

[2] 1394 standards and specifications summary, apr 2006. URL http://www.1394ta.org/developers/specifications/StandardsOrientationV5.0.pdf.

[3] Pci express base specification, nov 2010.

[4] Common vulnerability scoring system version 3.1: Specification document, jun 2019. URL https://www.first.org/cvss/specification-document.

[5] Intel processor support for vt-d, jan 2021. URL https://ark.intel.com/content/www/us/en/ark/search/featurefilter.html?productType=873&0_VTD=True.

[6] Benny Akesson. An introduction to sdram and memory controllers. 25:1–30, 2017.

[7] Amazon.com. Best sellers in computer servers, nov 2020. URL https://www.amazon.com/Best-Sellers-Electronics-Computer-Servers/zgbs/electronics/11036071.

[8] An-d. Swissbit 2gb pc2-5300u-555 ddr2 ram, jul 2013. URL https://commons.wikimedia.org/wiki/File:Swissbit_2GB_PC2-5300U-555.jpg#/media/File:Swissbit_2GB_PC2-5300U-555.jpg.

[9] Dan Lake Anna Trikalinou. Taking dma attacks to the next level: How to do arbitrary reads/writes in a live and unmodified system using a rogue memory controller, jul 2017. URL https://www.blackhat.com/docs/us-17/wednesday/us-17-Trikalinou-Taking-DMA-Attacks-To-The-Next-Level-How-To-Do-Arbitrary-Memory-Reac pdf.

[10] Humaira Arshad, Aman Bin Jantan, and Oludare Isaac Abiodun. Digital forensics: Review of issues in scientific validation of digital evidence. *Journal of Information Processing Systems*, 14 (2), 2018.

[11] Johannes Bauer, Michael Gruhn, and Felix C. Freiling. Lest we forget: Cold-boot attacks on scrambled ddr3 memory. *Digital Investigation*, 16:S65 – S74, 2016. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2016.01.009. URL http://www.sciencedirect.com/science/article/pii/S1742287616300032. DFRWS 2016 Europe.

[12] Vincent Nicomette Eric Alata Benoit Morgan, Guillaume Averlant. Bypassing dma remapping with dma, jun 2016. URL https://www.sstic.org/media/SSTIC2016/SSTIC-actes/dma_bypass_with_dma/SSTIC2016-Slides-dma_bypass_with_dma-morgan_alata_averlant_nicomette_1.pdf.

[13] Michael Berthold. Random data file creator (rdfc), dec 2004. URL http://www.bertel.de/software/rdfc/index-en.html.

[14] Bitdefender.com. How to generate a complete memory dump on windows 10, aug 2020. URL https://www.bitdefender.com/support/how-to-generate-a-complete-memory-dump-on-windows-10-1590.html.

[15] Erik-Oliver Blass and William Robertson. Tresor-hunt: Attacking cpu-bound encryption. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 71–78, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450313124. doi: 10.1145/2420950.2420961. URL https://doi.org/10.1145/2420950.2420961.

[16] Anthony Bonkoski, Russ Bielawski, and J Alex Halderman. Illuminating the security issues surrounding lights-out server management. In *7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13)*, 2013.

[17] Joe Grand Brian D. Carrier. A hardware-based memory acquisition procedure for digital investigations, feb 2004. URL `http://www.digital-evidence.org/papers/tribble-preprint.pdf`.

[18] Brian D. Carrier and Joe Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1(1):50–60, 2004. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2003.12.001. URL `https://www.sciencedirect.com/science/article/pii/S1742287603000021`.

[19] Eoghan Casey. Practical approaches to recovering encrypted digital evidence. *International Journal of Digital Evidence*, 2002.

[20] Bplus Tech. Corporation. Pe4h (pcie passive adapter ver2.4), mar 2021. URL `http://www.hwtools.net/Adapter/PE4H.html`.

[21] Intel Corporation. What is direct media interface (dmi)?, jun 2010. URL `https://community.intel.com/t5/Processors/What-is-Direct-Media-Interface-DMI/td-p/228957`.

[22] Intel Corporation. Intel® xeon® processor e5-1600/2400/2600/4600 (e5-product family) product families datasheet- volume two, may 2012. URL `https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v3-datasheet-vol-2.pdf`.

[23] Intel Corporation. Thunderbolt technology, mar 2021. URL `https://www.intel.com/content/www/us/en/products/docs/io/thunderbolt/thunderbolt-technology-general.html`.

[24] Franck Courbon, Sergei Skorobogatov, and Christopher Woods. Reverse engineering flash eeprom memories using scanning electron microscopy. pages 57–72, 03 2017. ISBN 978-3-319-54668-1. doi: 10.1007/978-3-319-54669-8_4.

[25] dan farmer. Ipmi: Freight train to hell, aug 2013. URL `http://fish2.com/ipmi/itrain.pdf`.

[26] Xiaoyu Du, Nhien-An Le-Khac, and Mark Scanlon. Evaluation of digital forensic process models with respect to digital forensics as a service, 2017.

[27] LamdaConcept EnjoyDigital. Screamer m.2 documentation, may 2020. URL `http://docs.lambdaconcept.com/screamer/index.html`.

[28] Andrea Fortuna. How to extract a ram dump from a running virtualbox machine, jun 2017. URL `https://www.andreafortuna.org/2017/06/23/how-to-extract-a-ram-dump-from-a-running-virtualbox-machine/`.

[29] Ulf Frisk. Pcileech, dec 2020. URL `https://github.com/ufrisk/LeechCore`.

[30] Ulf Frisk. Pcileech, dec 2020. URL `https://github.com/ufrisk/pcileech`.

[31] Ulf Frisk. Pcileech-fpga, dec 2020. URL `https://github.com/ufrisk/pcileech-fpga`.

[32] Pavel Gladyshev and Afrah Almansoori. Reliable acquisition of ram dumps from intel-based apple mac computers over firewire. In Ibrahim Baggili, editor, *Digital Forensics and Cyber Crime*, pages 55–64, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19513-6.

[33] Richard Gooch. Mtrr (memory type range register) control, jun 1998. URL `https://www.kernel.org/doc/Documentation/x86/mtrr.txt`.

[34] Miltos Grammatikakis, Kyprianos Papadimitriou, Polydoros Petrakis, Marcello Coppola, and Michael Soulie. Address interleaving for low-cost nocs. 06 2016. doi: 10.1109/ReCoSoC.2016.7533892.

[35] Gamma Group. Finfisher: Finfirewire 3.5 release notes, jan 2014. URL https://wikileaks.org/spyfiles/document/finfisher/finfirewire/Release-Notes-FinFireWire-3.5.pdf.

[36] Hilbert Hagedoorn. Amd ryzen 3000: New block diagram about pcie 4.0 on matisse and x570 chipset, may 2019. URL https://www.guru3d.com/news-story/amd-ryzen-3000-new-block-diagram-about-pcie-4-on-matisse-and-x570-chipset.html.

[37] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, may 2009. ISSN 0001-0782. doi: 10.1145/1506409.1506429. URL https://doi.org/10.1145/1506409.1506429.

[38] Peter Hannay and Andrew Woodward. Cold boot memory acquisition: An investigation into memory freezing and data retention claims. In *Security and Management*, pages 620–622, 2008.

[39] Compter Hope. Northbridge, jul 2019. URL https://www.computerhope.com/jargon/n/northbri.htm.

[40] Trammell Hudson. Thunderstrike 2, sep 2020. URL https://trmm.net/Thunderstrike_2/.

[41] Joseph Tartaro Ilja van Sprundel. Things not to do when using an iommu, oct 2020. URL https://www.youtube.com/watch?v=p1HUpSkHcZ0.

[42] Microchip Technology Inc. Pic32 family reference manual, jan 2013. URL http://ww1.microchip.com/downloads/en/DeviceDoc/60001214A.pdf.

[43] JEDEC. Main memory: Ddr4 & ddr5 sdram, jan 2021. URL https://www.jedec.org/category/technology-focus-area/main-memory-ddr3-ddr4-sdram.

[44] kernel.org. Pat (page attribute table), feb 2020. URL https://www.kernel.org/doc/html/latest/_sources/x86/pat.rst.txt.

[45] Guillaume Valadon Olivier Levillain Loic Duflot, Yves-Alexis Perez. Can you still trust your network card?, mar 2010. URL https://www.ssi.gouv.fr/uploads/IMG/pdf/csw-trustnetworkcard.pdf.

[46] Carsten Maartmann-Moe. Inception, mar 2021. URL https://github.com/carmaa/inception.

[47] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Internet Measurement Workshop (IMW)*, pages 273–284, Nov 2002.

[48] Tilo Müller, Felix C Freiling, and Andreas Dewald. Tresor runs encryption securely outside ram. In *USENIX Security Symposium*, volume 17, 2011.

[49] NateBrune. fmem - linux kernel module designed to help analyze volatile memory in the linux kernel, apr 2020. URL https://github.com/NateBrune/fmem.

[50] PCI-SIG. Pci-sig.com, nov 2020. URL https://pcisig.com/.

[51] Fabien Perigaud. Using your bmc as a dma device: Plugging pcileech to hpe ilo 4, dec 2018. URL https://www.synacktiv.com/en/publications/using-your-bmc-as-a-dma-device-plugging-pcileech-to-hpe-ilo-4.html.

[52] pinouts.ru. Pci express 1x, 4x, 8x, 16x bus pinout, oct 2020. URL https://pinouts.ru/Slots/pci_express_pinout.shtml.

[53] C. L. Rothwell. Exploitation from malicious pci express peripherals (doctoral thesis). 2018. doi: https://doi.org/10.17863/CAM.21474.

[54] Joanna Rutkowska. Beyond the cpu: Defeating hardware based ram acquisition. *Proceedings of BlackHat DC*, 2007, 2007.

[55] Björn Ruytenberg. Breaking thunderbolt protocol security: Vulnerability report. 2020. URL https://thunderspy.io/assets/reports/breaking-thunderbolt-security-bjorn-ruytenberg-20200417.pdf.

[56] Ruud Schramp. Ram memory acquisition using live-bios modification, aug 2013. URL https://www.youtube.com/watch?v=i_WvtO1NIsA.

[57] Johannes Stüttgen and Michael Cohen. Anti-forensic resilient memory acquisition. *Digital Investigation*, 10:S105–S115, 2013. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2013.06.012. URL https://www.sciencedirect.com/science/article/pii/S1742287613000583. The Proceedings of the Thirteenth Annual DFRWS Conference.

[58] Matt Suiche. Your favorite memory toolkit is back, sep 2016. URL https://blog.comae.io/your-favorite-memory-toolkit-is-back-f97072d33d5c.

[59] B. Gutstein A. Pearce P. Neumann S. Moore R. Watson T. Markettos, C. Rothwell. Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals. In *Network and Distributed System Security Symposium*, 2019. doi: https://doi.org/10.14722/ndss.2019.23194. URL https://thunderclap.io/thunderclap-paper-ndss2019.pdf.

[60] Rafal Wojtczuk Joanna Rutkowska Alexander Tereshkin. Another way to circumvent intel trusted execution technology, dec 2009. URL https://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf.

[61] Gregory Travis, Ed Balas, David Ripley, and Steven Wallace. Analysis of the “sql slammer” worm and its effects on indiana university and related institutions, 2004.

[62] Mark van Beusekom. Circumventing secure jtag: A detailed plan of attack. 2019.

[63] N van Heijningen. Professionalizing hardware-based memory acquisition for incident response scenarios cold boot using coreboot and the intel memory scrambler. 2017.

[64] Ministerie van Justitie en Veiligheid. *Voortgang integrale aanpak van cybercrime*. 2020.

[65] Olaf van Miltenburg. Microsoft koopt rootkitbestrijder, mar 2008. URL https://tweakers.net/nieuws/52549/microsoft-koopt-rootkitbestrijder.html.

[66] Victorgrigas. Server der wikimedia foundation, aug 2021. URL https://de.wikipedia.org/wiki/Server#/media/Datei:Wikimedia_Foundation_Servers-8055_35.jpg.

[67] Stefan Vömel and Felix C. Freiling. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digital Investigation*, 8(1):3–22, 2011. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2011.06.002. URL https://www.sciencedirect.com/science/article/pii/S1742287611000508.

[68] Stefan Vömel and Felix C. Freiling. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125 – 137, 2012. ISSN 1742-2876. doi: https://doi.org/10.1016/j.diin.2012.04.005. URL http://www.sciencedirect.com/science/article/pii/S1742287612000254.

[69] Wikipedia. List of iommu-supporting hardware, apr 2021. URL https://en.wikipedia.org/wiki/List_of_IOMMU-supporting_hardware.

[70] Alex Williamson. Intel processors with acs support, oct 2015. URL https://vfio.blogspot.com/2015/10/intel-processors-with-acs-support.html.

[71] Xilinx. 7 series fpgas integrated block for pci express v3.0 logicore ip product guide (pg054), nov 2014. URL https://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v3_0/pg054-7series-pcie.pdf.

[72] Xilinx. 7 series fpgas data sheet: Overview (ds180), dec 2020. URL `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`.