ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Bytes on the Loose - DMA Forensic Abuse

A Survey of Direct Memory Access in IT-Forensics

Masters Thesis

for attainment of the academic degree of

Diplom-Ingenieur/in

submitted by

Florian Nocker BSc

in the
University Course IT Security at St. Pölten University of Applied Sciences

Supervision
Advisor: FH-Prof. Dipl.-Ing. Dr. Robert Luh, BSc
Assistance: -

Declaration

Title: Bytes on the Loose - DMA Forensic Abuse
Type of Thesis: Masters Thesis
Author: Florian Nocker
Student number: is211814

I hereby affirm that

- I have written this thesis independently, have not used any sources or aids other than those indicated, and have not made use of any unauthorized assistance.
- I have not previously submitted this thesis topic to an assessor for evaluation or in any form as an examination paper, either in Austria or abroad.
- this thesis corresponds with the thesis assessed by the assessor.

I hereby declare that

☐ I have used a Large Language Model (LLM) to proofread the thesis.
☐ I have used a Large Language Model (LLM) to generate portions of the content of the thesis. I affirm that I have cited each generated sentence/paragraph with the original source. The LLM used is indicated by a footnote at the appropriate place.

☒ No Large Language Model (LLM) has been used for this work.

St. Pölten, 16.01.2024 (Signature Author)

Kurzfassung

Der direkte Speicherzugriff (Direct Memory Access bzw. DMA) ist ein grundlegender Bestandteil moderner Computerarchitekturen und eine Technik, die darauf abzielt, Datenübertragungen von der CPU auf andere Computerhardware, oft ein Peripheriegerät, auszulagern. Dies führt dazu, dass mehr CPU-Ressourcen für andere Aufgaben zur Verfügung stehen, während Peripheriegeräte vollen Zugriff auf den Systemspeicher erhalten. Diese Sicherheitslücke kann für die forensische Speicherakquise von Computersystemen genutzt werden.

Im Rahmen unserer Arbeit haben wir den aktuellen Wissensstand über die Ausnutzung von DMA und Gegenmaßnahmen im Kontext der digitalen Forensik untersucht. Mittels einer strukturierten Literaturrecherche geben wir einen umfassenden Überblick über den Stand der Technik bei DMA-Angriffen in forensisch relevanten Anwendungsszenarien sowie über etablierte Abhilfemaßnahmen. Die meisten neuen Ansätze in diesem Bereich erfordern umfangreiche Kenntnisse über verschiedene verwandte Technologien und Datenübertragungsmechanismen. Unsere Arbeit soll einen Beitrag zum Verständnis von DMA, seinen Ausnutzungsmöglichkeiten für den forensischen Zugriff auf den Systemspeicher und benachbarte Technologien leisten, die eng damit verwandt oder davon abhängig sind.

Abstract

Direct Memory Access (DMA) is a fundamental component of computer architecture and is a technique aimed to offload data transfers from the CPU onto different computing hardware, often times a peripheral device. This results in more CPU resources being available for other tasks, at the expense of granting peripheral devices full access to system memory. This security flaw can be used for forensic memory acquisition of computer systems.

During our work, we examined the knowledge body of DMA exploitation and - countermeasures in the context of digital forensics. Via the conduction of a structured literature review, we provide a comprehensive survey of the state-of-the-art for DMA attacks in forensically relevant scenarios as well as established mitigation techniques. Most novel approaches in this field require extensive knowledge on various related technologies and data transfer mechanisms. Our deliverable shall contribute towards the understanding of DMA, its ways of exploitation to gain forensic access to system memory and adjacent technologies closely related to- or dependent on it.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor Mr. Robert Luh for their encouraging words of support, their feedback as well as their understanding for the challenges and circumstances the process of writing this thesis has been associated with.

Additionally I shall extend my gratitude to my dear friends Mr. Bernhard Rader and Mr. Marius Nesch - Thank you for being my highly regarded class mates, work colleagues, roommates and family. I wouldn’t have made it this far without *The Bois*.

Table of Contents

Introduction

Direct Memory Access (DMA) is a fundamental component of computer architecture and is a technique aimed to offload data transfers from the CPU onto different computing hardware, often times a peripheral device [1]. This results in more CPU resources being available for other tasks, as it is no longer required to mediate the data transfers.

On a basic level, this works by assigning specific areas of the system memory to a given device, thus allowing the device to read and write directly to and from the memory. [2, p. 122, 123]

Although DMA is a useful feature leveraged by the vast majority of modern computing- and bus architectures [1], its ability to interact with the system memory in a way that is transparent for the host CPU make it an inherent weakness from an IT-Security perspective. This vulnerability thus opens up possible ways of exploitation where an unauthorized device may attain direct access to the system's memory whilst bypassing traditional security controls such as the need for correct logon-credentials.

Simply put, DMA exploitation can be used as a means of gaining access to a locked computer system and obtaining sensitive data such as the following [3]:

- passwords
- encryption keys (which allows circumvention of disk encryption)
- the operating system residing in memory
- recently connected devices
- executed code of any kind
- traces of malware

Due to the nature of the extractable data, interested parties[1] in this kind of attack technique include penetration-testers, IT-forensic investigators (especially as part of law-enforcement bodies), security researchers as well as blue-teaming- and incident-response departments.

[1]Apart from maliciously motivated entities and threat-actors

Even though the attack vector DMA on system memory content has been known by system vendors and -manufacturers such as Apple [4] and Microsoft [5] for a significant amount of time, several researchers and publications have demonstrated ways to bypass corresponding protection mechanisms as has been shown in Section 4.1.

The authors believe that providing a survey of the current state-of-the-art regarding DMA, attack techniques based on it as well as corresponding security controls pertaining to it poses a valuable contribution.

This is due to the fact that most novel approaches to DMA exploitation in IT-Security require a comprehensive understanding of the technologies involved as well as an overview of the pre-existing attack vectors for DMA. Reviewing the different publications detailing sophisticated ways of abusing DMA access for a certain purpose reinforced the need for such a comprehensive summary of the subject at hand.

It should also be noted that on an individual level, the thesis authors' motivation for this literature review has also been aroused by their previous work on forensic memory acquisition using an FPGA-based attack platform [6].

The remainder of this thesis is structured as follows:

Following this introduction, the background section provides necessary know-how on DMA and its basic functionality, related technologies and mechanisms. Subsequently, the methodological approach to the conducted research in addition to its limitations and relevant contributions are discussed in Section 3.

After Section 4 surveys the corresponding knowledge domains about DMA and its role in IT-Forensics and -security as per the research objectives defined in Section 3.1, its results are examined, and a taxonomy of the inspected source material is discussed in Section 5.

Finally, Section 5 elaborates on potential approaches to continuing this work specifically as well as applicable future work pertaining DMA in the context of IT-security in general.

Background

This section should provide a concise review of the necessary background regarding DMA fundamentals, its ways of transferring data and how it compares to different means of I/O in a computing system.

Subsequently, related mechanisms on modern PCI(e) bus systems, memory mapping of devices and corresponding memory address spaces are discussed before examining an actual DMA data transfer in-depth.

DMA Basics

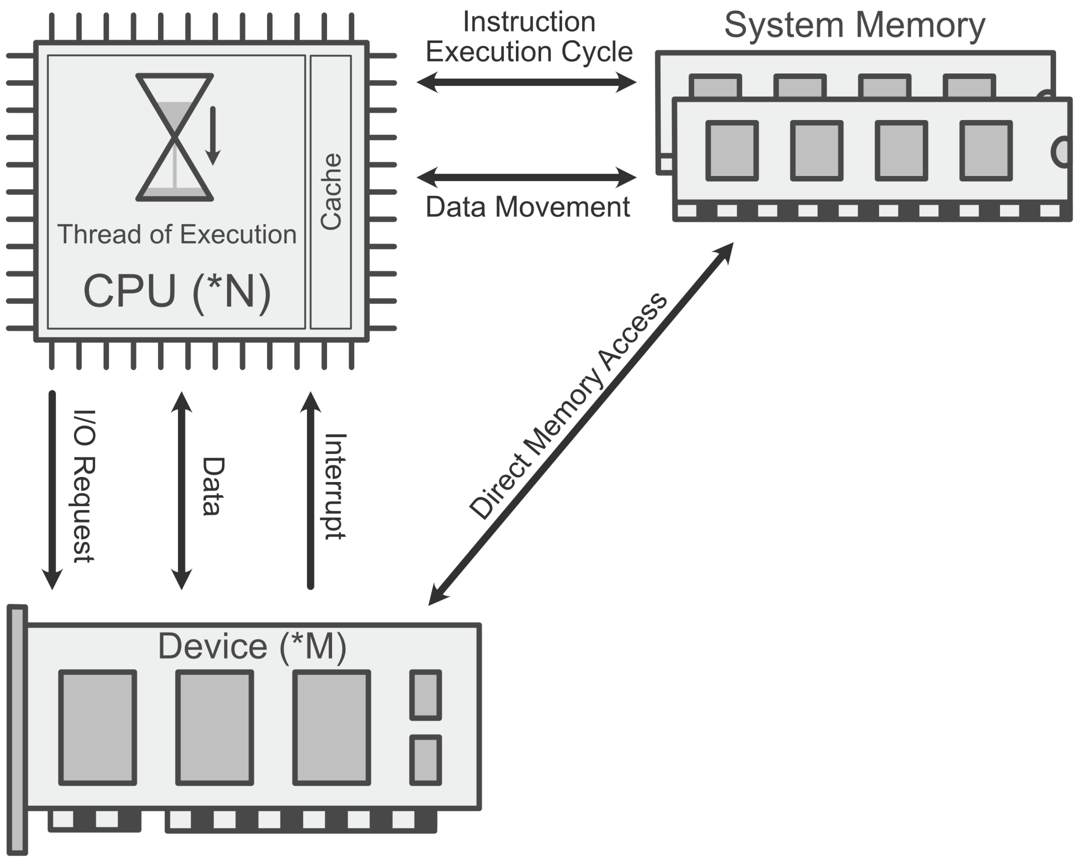

Figure 1: Illustration of DMA in Contrast to alt. Data Transfer Methods, Source: [7]

General Data Transfer Mechanisms

Figure 1 graphically distinguishes the three primary means of transferring data [1] in modern computer systems:

- Polling I/O
- Interrupt-driven I/O
- DMA

In contrary to the latter, polling I/O [8] refers to the process of actively sampling a data source such as a device by software code in order to retrieve data. The cyclical occurrence of such polls is often controlled by timers, such as hardware-based watchdog-timers [9].
As the name implies, in interrupt-driven I/O [10] the actual data transfer operation is performed after the computing unit has encountered an interrupt, a request to the processor to temporarily suspend its current execution task and process the event-code entailed by the interrupt routine. This saves computing resources in comparison to polling I/O by not having to sample the state of the data source constantly.

DMA however achieves full CPU-offloading by executing the necessary instructions to transfer data on an entirely different computing component [1], freeing up a significant amount of processing resources for other, more important activities. For this reason, DMA has become a fundamental component of the digital systems we use today.

First- vs. Third-Party DMA

When addressing the topic of DMA, the distinction between first- and third-party DMA needs to be emphasized.
The former relies on a device[2] to use a built-in DMA engine in order to perform the I/O action.
The latter describes DMA for devices that do not implement such an engine themselves, but rather have a central DMA controller orchestrate and perform the data transfer, which is common for legacy devices such as Industry Standard Architecture (ISA) devices [11, p. 16].

[2]In the context of a personal computer or server, device here refers to external devices such as graphics cards, I/O controllers, display controllers etc.

PCI(e) Bus System Deep-Dive

As per Section 3, this thesis focus lies on personal computers and servers due to them being the most common target for forensic use cases. Nowadays, the underlying bus system connecting the components, devices and interfaces (e.g. Thunderbolt [12]) of these systems internally is the PCI Express bus.
Therefore, the required background pertaining these bus systems needs to be examined in order to understand how the DMA mechanisms work on a technical level.

PCI(e) Basics and Terminology

The originally defined PCI standard is considered legacy today, but reviewing its properties helps derive differences in implementation for its successor PCIe as well as provides a fundamental understanding of bus communication concepts. Thus, it shall be discussed briefly alongside its modern derivative.

For instance, due to its *shared-line property*, PCI distinguishes between bus masters and bus agents. A bus master is a participating device on the bus that is capable of initiating bus transactions (e.g. a data transfer to or from system memory). However not every device operating on the bus necessarily has to be a bus master, but rather a bus agent [13, p. 13]. Simply speaking, a bus master is in control of the bus while a bus agent is a participant on the bus.
Since bus systems are usually shared between multiple agents (or even multiple masters, as is the case for PCI), the need for a mechanism to determine which device may use the bus at a given time emerges. This is used to avoid bus collisions[3] and is referred to as *bus arbitration*.
In PCI, bus masters need to request ownership of the bus prior to performing transactions on it [13, p. 15]. These requests are being processed by the bus arbiter and bus ownership is granted accordingly [13, p. 59].

PCIe set itself apart from its predecessor PCI by introducing multiple lanes, each consisting of a transmit- and receive pair, together forming a *dual-simplex channel* [14].

[3] A bus collision happens when more than one device sends data on the bus, hence interfering with each other's communication and rendering the sent data unrecoverable.

Albeit being introduced for bandwidth increase and the performance gains resulting thereof, this change also superseded the shared-line concept with a *point-to-point property*. For this reason, PCIe requests do not need to be arbitrated [11, p. 17].

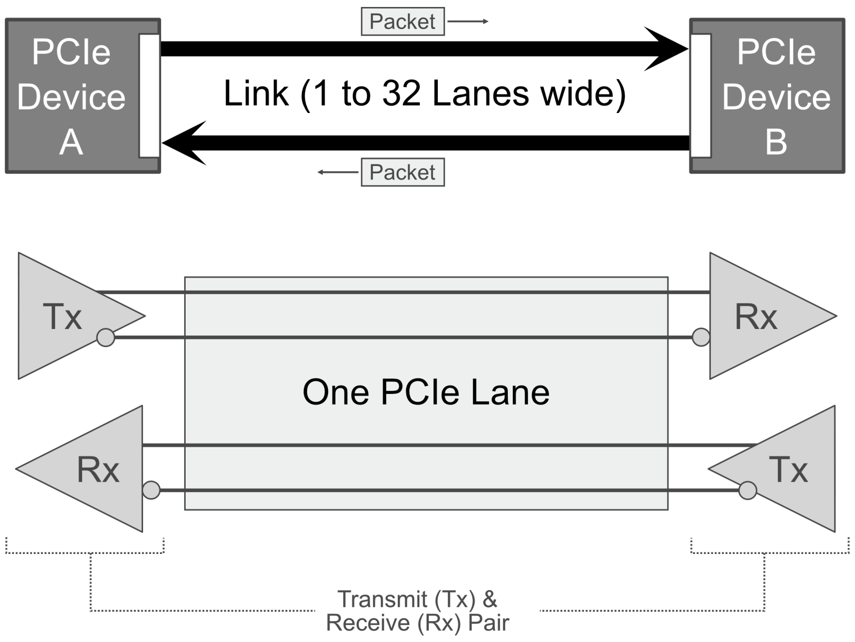

Figure 2: Link and Lane Relationship in PCIe, Source: [2]

Jackson et al. [2, p. 40] mention that the spec for the PCIe standard describes the communication channel between PCIe devices as dual-simplex, but it is actually capable of full duplex communication[4].
The path between devices as illustrated in the upper diagram of Figure 2 is called a *PCIe link* and consists of one or more transmit- and recieve pairs. These pairs are called a *PCIe lane* and the specification allows for links to be made up of 2^n number of lanes up to x32, improving the available bandwidth on the link as well as power consumption and space requirement on the PCB (Printed Circuit Board).

As has been demonstrated by the Author's previous work [6], attaining a logical PCIe Link (as seen by the OS) to a malicious or compromised device is a crucial step in the majority of DMA exploitation techniques[5]. Being able to achieve this state without compromising the target system's integrity and memory content is indicative whether an attack technique is relevant for forensic use cases.

[4]The channel has a pair of one-way lines to communicate in both "directions" but the devices are actually capable of communicating in both directions at the same time.
[5]The device needs to be enumerated on the bus, thus granting a logical link ID that is required to perform transactions on the PCIe bus.

PCI(e) Memory Address Spaces

To ensure compatibility with its predecessor technology, PCIe supports the same (memory-) address spaces as PCI [2, p. 122]:

- Configuration Address Space
- I/O Address Space
- (Memory Mapped I/O)

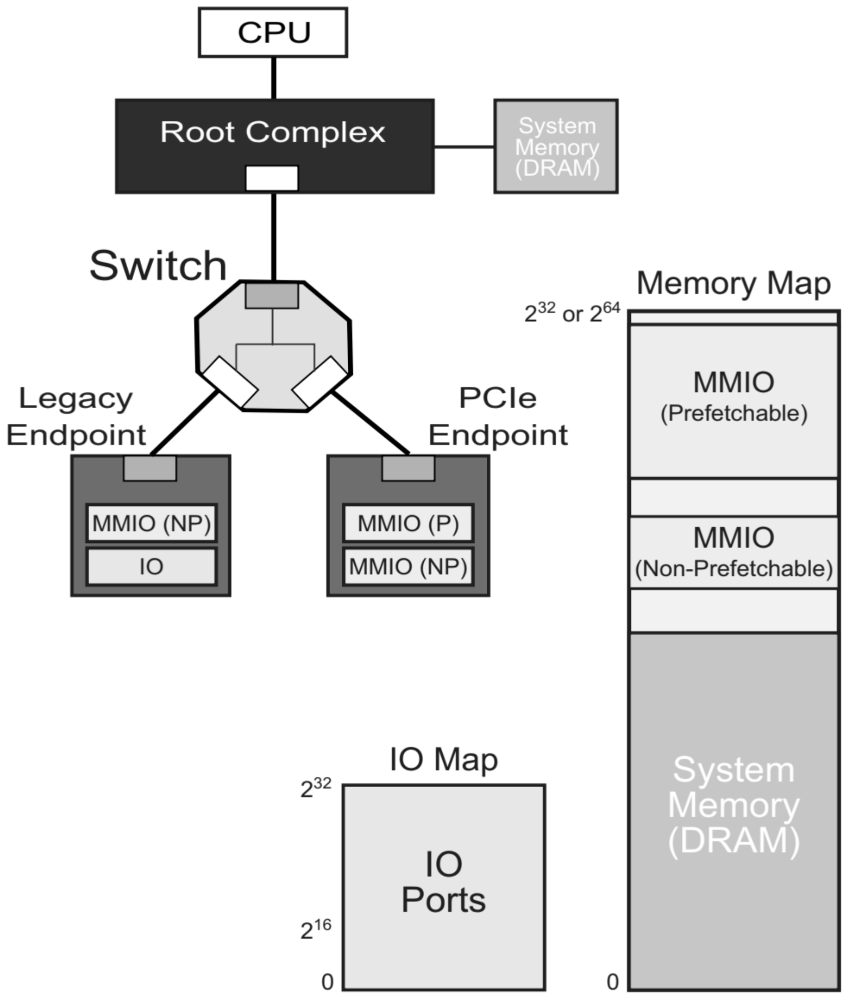

Figure 3: Overview of Generic Memory- and I/O Address Maps, Source: [2]

PCIe devices may have their internal registers, buffers and storage mapped into the corresponding locations inside the system memory [2, p. 125] as depicted in Figure 3.

A PCIe device might provide more than a single functionality (e.g. a storage controller may implement a USB controller alongside its main functionality etc.) and subsequently consists of one or more *functions* (logical devices) [2, p. 86].
Therefore, each function requires its own configuration register [2, p. 310-311]. These registers are mapped to a location in the system memory called the *Configuration Address Space*. This memory location is used for identification, status polling and configuration of devices and their functions [2, p. 88, 122]. It should be mentioned that each logical device has its own configuration address space.

In a very similar fashion, the *I/O Address Space* [2, p. 122] maps the internal storage and registers of I/O devices. Isolated I/O used to be the standard way for accessing them as has been defined by the now legacy standard from Intel Corporation.

Due to its limitations, soft- and hardware manufacturers have deprecated this mechanism in favor of mapping the storage and registers of devices to the *Memory Address Space*. This technique is referred to as *Memory Mapped I/O* or MMIO for short. Even though I/O address space mapping is widely considered legacy tech today, it is still often implemented alongside MMIO by controllers to ensure backwards compatibility [15] [2, p. 123].

In general, the process of memory mapping acts as an abstraction layer for system code that needs to access registers or I/O ports of devices[6]. Due to this abstraction layer [16, ch. 19], writing to an I/O port, register or buffer of a device is analogue to writing to a certain memory location. Thus, MMIO does not require any special consideration from the processor, in turn meaning that all processors are capable of MMIO.
The I/O address space on the other hand is separate from the memory address space (demonstrated in Figure 3) and for this reason special I/O instructions are required to interact with it.

[6]Function registers are actual hardware registers inside the PCIe device

PCIe Bus Architecture

Before diving into how DMA works on PCIe busses, the basics of PCIe bus architecture need to be reviewed.
Figure 4 shows an example PCIe topology. The root of the hierarchical topology is the *Root Complex* (RC) that is directly connected to the CPU and memory controller. On modern mainboards, the RC is integrated into the chipset as opposed to being a standalone integrated-circuit and connects processor as well as system memory to the I/O subsystem [17].
The RC may be assumed a host- or north bridge equivalent [18]. It communicates with the rest of the computer system on behalf of the CPU and contains multiple components, for example processor- and memory interfaces.

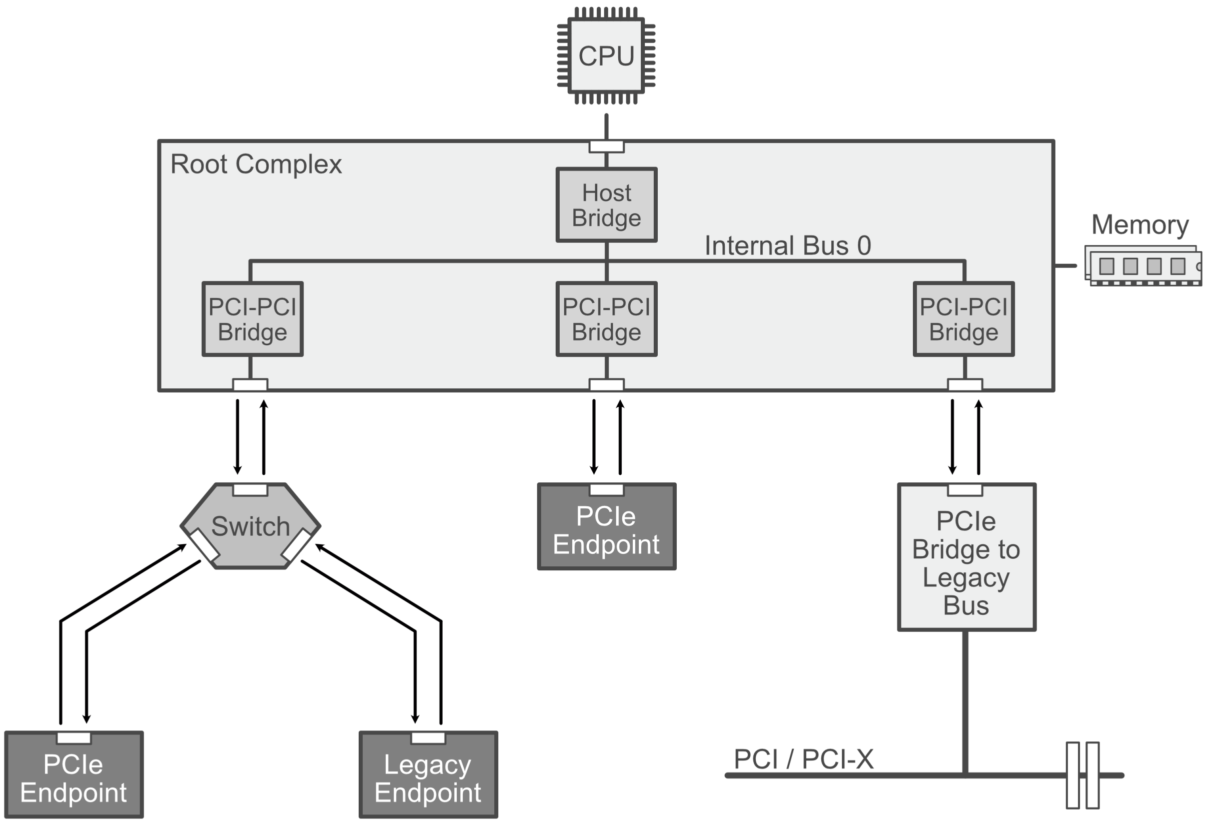

Figure 4: Various PCIe Topology Components, Source: [2]

PCIe Switches provide an aggregation capability by allowing more devices or endpoints to be attached to a single PCIe Port [2, p. 48]. They are also responsible for making the routing decisions necessary for PCIe Transport Layer Packets (TLP) [17] to reach their destination.

PCIe Endpoints refer to the PCIe devices connected to the bus topology. Native endpoints should be distinguished from legacy endpoints in the fact that they have been designed for use with the PCIe standard [2, p. 49] as opposed to one of its precursors (e.g. PCI or PCI-X). Although Legacy endpoints may be operable on a PCIe bus, they do not necessarily have to and are likely to rely on deprecated technologies and mechanisms such as programmed I/O via the I/O address space as discussed prior.
Endpoints can assume the role of a requester or completer [17], depending on whether they initiated a transaction on the bus or are responding to a received transaction.

DMA Data Transfer in PCIe Bus Systems

Now that the PCI(e) bus systems and their properties have been examined closely, a DMA data transfer shall be distinguished from previously mentioned means of data transfers and then discussed step by step. It should be emphasized that such transactions are performed without any involvement of the CPU. It is not aware of the data transfer happening as opposed to its alternatives [19]:

	Interrupt-driven I/O	**Programmed I/O**
1.	I/O module of device fetches data	CPU requests I/O operation
2.	I/O module interrupts CPU	I/O module performs operation
3.	CPU requests data from device	I/O module sets status registers
4.	I/O module transfers data	CPU periodically checks status

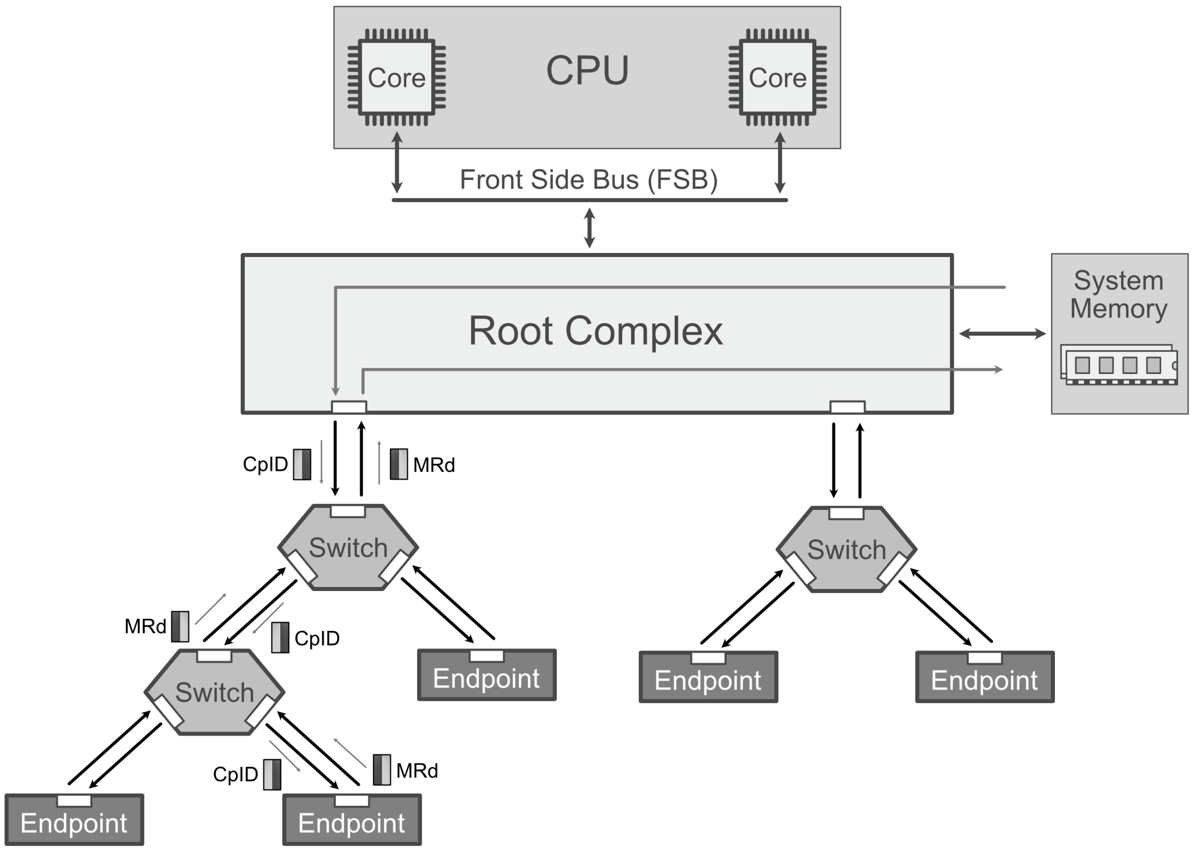

Figure 5: Execution of a DMA Transacion for a Device reading Memory, Source: [20]

For this scenario, a data target in system memory is assumed, this may either be main memory content or an MMIO location. Due to the DMA mechanism, such targets are not only accessible to the CPU, but also I/O controllers of PCIe endpoints [15].

Figure 5 illustrates the process of a peripheral device initiating a DMA transaction to read data from memory as described by R. Solomon [20]:

1. Endpoint initiates a Memory Read Request (MRd), therefore acting as the requester.
2. Root complex receives the MRd, it acts as the according completer for this transaction
3. RC acts on behalf of the CPU and fetches data from memory, responding to the endpoint with Completion with Data (CplD)
4. Endpoint receives requested data (CplD)

In conclusion, this means that any PCIe device capable of performing DMA transactions on the bus is able to *read and write arbitrary memory locations*. This entails significant consequences for the security of any computing system implementing DMA.

Therefore, all DMA attacks rely on successfully carrying out a malicious memory access. As concisely put by Sang et al. [15], this allows to distinguish multiple ways to initiate such a malicious DMA transaction:

- Using the DMA engine programming interface[7] in order to instruct the I/O controller of a device to perform the memory access
- Exploiting a vulnerability in an I/O controller of a device to control its DMA engine from the I/O controllers side
- Building a custom I/O controller (e.g. introducing a malicious device into the target system)

[7]An analogue to an API for configuring and programming DMA engines

Methodology

In this section, the overarching methodological approach to the conducted research is outlined, going over the research-design, research-objective as well as covering the contributions and limitations associated with this work.

Research Design

The methodological approach to this thesis consists of qualitative research employing a *systematic literature review* [21]. This is due to the fact that this thesis aims to synthesize the current body of knowledge regarding *DMA and its exploitation in the context of IT-Security, especially Digital Forensics.*

In order to provide this synthesis, the current state of literature for this subject is thoroughly reviewed, interpreted and analyzed using the proposed research method [22] and complying to the rules and procedures defined as part of the methodological approach.

Scope

The scope of the conducted research confines itself to the structured examination of scholarly articles and papers, conference proceedings as well as technical reports and documentation. To elaborate further, the examined resources shall discuss DMA in a context of IT-Security and -forensics.

The survey covers literature published from the year 2004[8] up to the present to ensure all relevant information is taken into account for the review process.

Studies and resources examining DMA in absence of any IT-Security context or those that concern themselves with Direct Memory Access itself and its technical use-cases without substantive discussion regarding this context are not considered in this review. Furthermore, this survey shall especially put a focus on the vulnerabilities- and exploitation of DMA mechanisms in forensically relevant scenarios as well as common circumvention measures against such techniques. Finally, this review shall also provide an outlook on potential future trends and developments regarding the subject.

[8]time when first "academic attention" for DMA in conjunction with IT-Security emerged: [23]

Research Question

Since Direct Memory Access is a very broad field, the research objectives for this review are threefold:

- Summarizing literature on *functionality and properties of DMA* and its interaction with other system components on modern computers as well as providing the necessary background on other computing techniques closely related to- or dependent on it
- Surveying of *DMA vulnerabilities and their exploitation*, especially as part of a digital forensics use-case or generally relevant to IT-Security
- Exploring state-of-the-art *protection measures against DMA attacks* brought in place by system vendors and manufacturers

In order to achieve the research objectives laid out and provide a comprehensive overview of the current applications of DMA within these fields, the following research question has been formulated for this study:

What are the existing techniques for- and methods against exploiting Direct Memory Access (DMA) vulnerabilities in IT-Forensic use-cases?

Data Sampling and Processing

In terms of search strategy, a comprehensive literature review is conducted to identify relevant resources related to DMA in IT-Security and -forensics. The search is conducted using well established databases such as the following:

- IEEE Xplore
- Google Scholar
- ScienceDirect
- ACM Digital Library

As mentioned previously, the different types of resources that are reviewed consist of research articles, conference proceedings, technical documentation and -reports as well as relevant books and other publications in either print or digital form.

The criteria and procedure for the selection of literature is discussed as follows.

Selection of Literature

To establish a standardized way of selecting literature, a *dual-stage screening process* is applied. The first step involves screening titles and abstracts of the retrieved resources against the inclusion criteria:

- Publishing date from 2004 to the present, to ensure the relevancy of the technology discussed
- Language may either be English or German
- Resource must strictly focus on DMA in the context of IT-Security or -Forensics
- DMA vulnerabilities or exploits must relate to Personal- or Server Computing Systems (i.e. No DMA literature that focuses on VM-, mobile-, embedded-, IoT-, OT- or other unrelated technologies)

For all publications that are determined relevant according to this catalog of criteria, the second stage of the screening process is applied. This entails a full-text review of the resources, in turn evaluating based on their contribution to this thesis' research question. Publications that failed to provide adequate information to contribute towards achieving the research objectives are excluded.

Data Extraction and Analysis

Information is extracted from each of the selected publications including the author(s), publication date, research objectives of the study, the methodology used, key findings as well as their implications for IT-Security and digital forensics.

The measures employed to ensure an adequate level of data management focus on aggregating the collected literature in a central database. The open-source literature management appliance *Zotero* [24] has been chosen for this purpose.

A thematic analysis is conducted on the extracted data. This consists of identification, analysis and thematic categorization according to the research objectives discussed in Section 3.1.1.

Quality Assessment

Each retrieved resource was checked for quality and rigor. This validation procedure incorporated the clarity and relevancy of the research objectives and how appropriate the research design is as a means of achieving the research objectives. Furthermore, this process includes evaluating the validity of the findings and the quality and degree to which they were discussed:

Category	**Score Range**	**Abbreviation**
Research Objectives	[1 ; 3]	RO
Research Design	[1 ; 3]	RD
Findings & Discussion	[1 ; 3]	FD

Listing 1: Scoring Categories for the Quality Assessment of Literature

According to the proposed scheme in Listing 1, a literature resource can achieve a Quality Score of up to 9:

Depending on the degree to which the research objectives of an examined resource are relevant to fulfilling the research question of this thesis, a corresponding score is given for partly relevant (RO = 1), mostly relevant (RO = 2) and fully relevant (RO = 3) publications.

Depending on the degree to which the research design of an examined resource is suitable to fulfilling its research objectives, a corresponding score is given for mostly suitable (RD = 1), fully suitable (RD = 2) and fully suitable and beyond (RD = 3) designs.

Depending on whether the conducted research produced relevant results (FD = 1), resulted in relevant and novel findings (FD = 2) or concluded with relevant and novel findings with comprehensive discussion and contextual embedding (FD = 3), the FD score is assigned accordingly.

Limitation

The proposed methodological approach for the conducted literature research is subject to certain limitations.

To begin with, literature published in an alternate language or earlier date than what has been defined as respective scope-boundaries in Section 3.1.1 may be excluded. Following that, it should be emphasized that literature concerning itself with DMA in IT-Security may still not be included in this review depending on the host-technology it relies on. This is due to the wide range of use-cases for DMA as a computing mechanism. As has been explained in Section 3.2.1, only workstation (personal computing) and server computing platforms as the underlying host-technology are relevant.

Secondly, the quality assessment and selection procedure established throughout the research design may introduce bias trough the exclusion of certain publications. Additionally, despite the author's best efforts, the data sampling strategy may not have identified all relevant resources regarding the topic.

Finally, this study is limited to the search-engines or -databases as listed in Section 3.2. This is due to the fact that they are readily and freely available for academic participants of any kind.
In general, this literature review has been conducted relying solely on freely available resources or those provided to all students by FH St. Pölten (e.g. library services). All paid resources and appliances have been funded privately by the study author - For this reason, the entirety of this thesis is financially independent.

Contribution

This literature review is expected to provide a comprehensive understanding of the state-of-the-art in DMA exploitation- and protection techniques. For this reason, a survey of the current literature is delivered that aims to achieve the research objectives defined in Section 3.1.2 and ultimately provide a satisfactory answer to the formulated research question.

Therefore, the delivered survey shall provide a valuable source of condensed information regarding the subject for all interested parties, especially researchers, IT-Forensic investigators as well as IT-Security specialists. The deliverable shall also contribute to the broader understanding of the role and significance of DMA within the fields of IT Security and digital forensics.

Literature Review

According to the research issue set out for this review, this section is going to focus on different ways of exploiting DMA, especially publications that aim to use such attacks in IT forensic scenarios. To further satisfy the corresponding research objectives for this thesis, circumvention implementations for DMA exploits on operating system level distinguished by OS vendor are going to be discussed subsequently, before examining the underlying hardware platform components that lie at the core of the implementation stack that allows minimizing the impact of such attacks or even prevent them altogether.

DMA Exploitation

In 2004 and 2005, M. Dornseif et al. aroused academic interest in DMA exploitation with their conference presentations [23] [25]. Their work highlighted the vulnerabilities and forensic implications of the *FireWire* interface and presented an example attack conducted from an iPod as the source device.
FireWire is a serial bus technology developed by Apple, Texas Instruments and Sony and standardized as IEEE 1394 in 1995. Nowadays, it may be considered legacy tech that is only rarely encountered in modern computing systems as it has mostly been superseded by other technologies (e.g. ThunderBolt and modern USB revisions). FireWire used to be a widespread interface, encompassing printers, audio equipment and mobile devices among many others, especially prominent in the Apple ecosystem and for real-time applications due to its guaranteed bandwidth property via isochronous data transfer technology [26]. Figure 6 exhibits how the FireWire architecture consists of locally addressable node-entities.

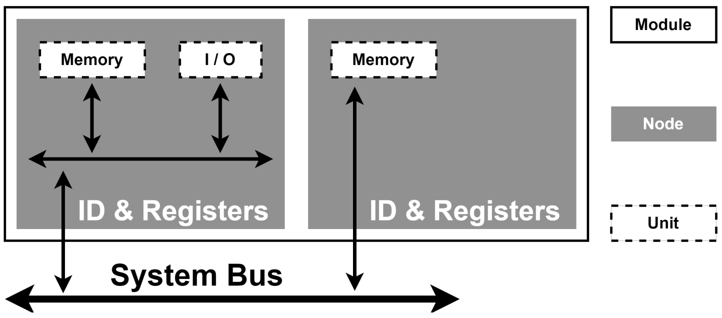

Figure 6: Node/Module Architecture for IEEE 1394, Source: [27, p. 847]

Sharing similarities with how PCIe endpoints are structured into functions, a module may contain multiple nodes that have their own unique address and control registers. A general topology overview as per Figure 7 displays how the FireWire serial bus is divided into a backplane- and cable environment.

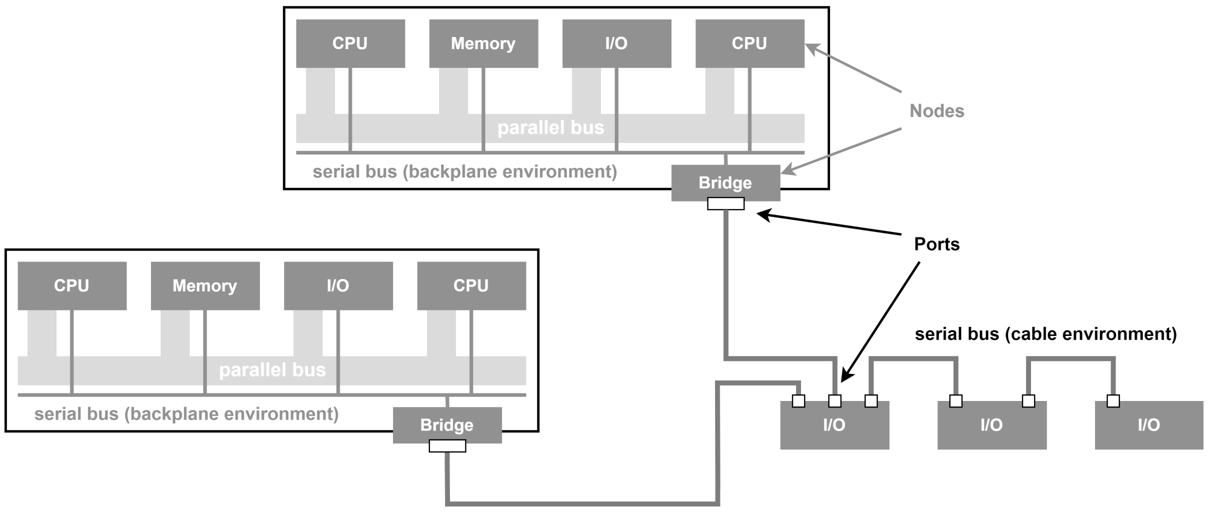
CPU
Memory
I/O
CPU
parallel bus
serial bus (backplane environment)
Bridge
Nodes
Ports
CPU
Memory
I/O
CPU
parallel bus
serial bus (backplane environment)
Bridge
serial bus (cable environment)
I/O
I/O
I/O

Figure 7: A High-Level Topology for IEEE 1394, Source: [27, p. 848]

Today it is likely still encountered in OT environments such as industrial control systems and therefore still retains a degree of relevancy. The authors explain the memory space of FireWire and deduct how the Open Host Controller Interface (OHCI) standard allows accessing RAM, acting as a proxy between memory and the requesting node much like the RC in PCIe, therefore granting full system access to malicious devices.

Dornseif et al. further solidify their findings by demonstrating comprehensive ways to utilize this property for exploits by executing arbitrary code and extracting key material from RAM. In depth implementation details for different operating systems are provided [25]:

- MacOS X
- Linux
- FreeBSD
- Windows 2000 and XP

They also cover forensic applications of abusing FireWire access to memory - They propose using the 1394/FireWire bus for live memory dumps as a solution for forensic acquisition of RAM content. The challenges associated with this use-case such as reconstructing virtual memory and correctly parsing data structures (e.g. OS kernel) in physical memory dumps are also mentioned.

Continuing with FireWire, A. Boileau shared further insights into using the IEEE 1394 and other bus systems to attain illicit memory access [26]. Their lead argument is that general purpose computers are not operable securely in hostile environments, as opposed to systems designed to be deployed in exposure to such threat models (e.g. ATM's, vending machines etc.). They distinguish expansion-bus systems from peripheral bus systems:

Expansion Bus

- PCI(e) / ISA
- PCMCIA / Cardbus
- Firewire IEEE 1394

Peripheral Bus

- Serial- / Parallel Port
- PS2/AT
- USB

The author emphasized that since expansion busses have direct bus-access as opposed to being abstracted by an intermediate protocol, which is the case for the peripheral busses, they also have unrestricted memory access. For this reason, the expansion bus systems also serve as DMA attack vectors.

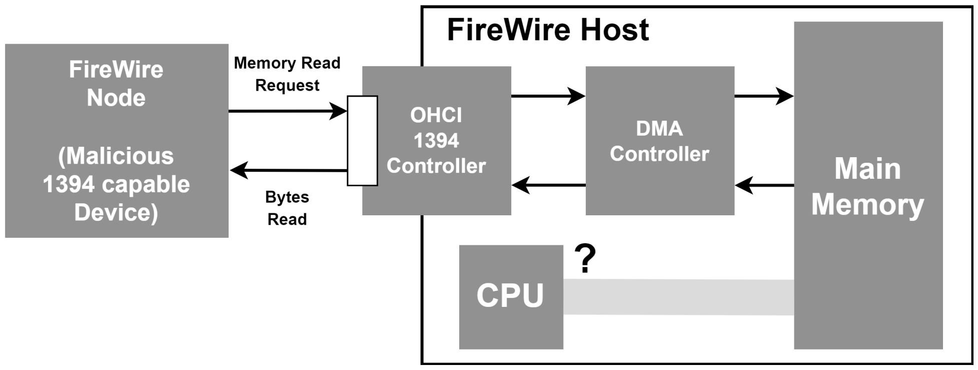

Figure 8: The DMA Transaction Process via IEEE 1394, Source: [26]

The demonstrated attack, depicted in Figure 8, targeted a Windows XPSP2 host (Boileau also successfully attacked the operating systems targeted by Dornseif et al.) and was able to succeed by impersonating a DMA-permitted device via setting the correct bits in the Configuration Status Registers (CSR) of the FireWire Node.

The author translates their findings into implications for forensic imaging of system memory, emphasizing that acquisition via FireWire is an attractive alternative to expensive hardware devices due to its ease of use, widespread usage of the interface and affordability aspect.

For Limitations, the reliability of IEEE 1394 based acquisition methods in terms of system crashes is mentioned in addition to the significant amount of manual work required to extract meaningful data from physical memory dumps and the immaturity of tooling for this purpose (This is no longer relevant today, as there is an abundance of tools and techniques available).
For mitigation strategies, only physical access restriction to- and (permanent) disabling of FireWire interfaces are proposed, referencing that because of its limitations, OHCI cannot properly facilitate mitigation extensions.

Boileau later released their custom written tool *winlockpwn* that they employed in their attack to bypass user credential authentication and spawn an administrator shell prior to user logon [28].

This tool has been improved upon by F. Witherden [29] as part of their insightful publication on memory forensics over the FireWire interface. They comprehensively explored the anatomy of the 1394 interface, its address spaces and DMA capabilities in the context of IT-forensic acquisition of system memory.

Witherden argues that the high degree of availability for the FireWire interface makes it an ideal enabler for forensic memory access. If the target system does not implement FireWire natively (which is to be expected on modern systems nowadays), it may be supplied via expansion cards:

Desktop Systems

- PCI(e)
- M.2 Interface
- Thunderbolt

Notebooks

- ExpressCard
- PC Card
- Thunderbolt

Adding to that, they state that the *hot plug capability* of certain interfaces such as PC Card and ExpressCard (Today also Thunderbolt and in some cases even PCI(e)[9]) allows adding a 1394 interface to the target system while it is running.
This is very useful, as it enables forensic memory acquisition in a large number of threat models, for example when encountering a locked running system that needs to be acquired, but may not be rebooted, lest the memory content is lost.

[9]As per our previous research, it has been shown that PCI(e) hot plug capability is firmware-, hardware- and OS implementation specific [6].

Witherden also composed a list of criteria to measure the quality of an acquired memory image [29]:

- *Completeness:* Percentage of relevant data captured (main memory, memory mapped locations, caches etc.)
- *Consistency:* Amount of inconsistencies in the acquired image due to the ever-changing nature of volatile memory increases with required time for image acquisition.
- *Reliability:* Images should be free from tampering (requiring to run software on a target host introduces changes into the image). Hardware acquisition methods are usually less invasive or free of tampering altogether.
- *Tool Marks:* Image tampering by introducing the acquisition tool or -method as well as memory degradation induced by the imaging process.
- *Risk Factor:* Some acquisition methods have a chance of failure with consequences ranging from introducing additional image inconsistencies up to complete loss of memory contents by crashing the target system.

The study author then compared DMA-based 1394 acquisition to well-established alternative acquisition methods suitable for desktop- and notebook systems:

Characteristic	**Hardware**		**Software**			
	1394	Other	Hiberfile	Live	Hot Boot	Cold Boot
Prerequisites	✓	××	✓	×	✓	××
Completeness	✓	✓✓	×	✓✓	×	×
Consistency	×	×	✓	×	✓✓	✓✓
Reliability	×	✓	×	××	✓	✓✓
Tool Marks	✓	✓	✓	××	✓	✓
Degradation	✓✓	✓✓	✓✓	✓✓	✓	×
Risk Factor	✓	×	✓✓	✓	××	××

✓✓ = very good ✓ = good × = poor ×× = very poor

Listing 2: Comparison of Memory Acquisition Methods

Because of its broad availability, hot plug capable deployment options, cost-effective and low risk usage scenarios as demonstrated in Listing 2, the authors deemed 1394 a very promising memory imaging method.
Its usefulness in today's application scenarios is heavily stunted by its limitation to

low address space (the first 4 GB of memory). Theoretically, the OHCI standard allows for a `PhysicalUpperBound` register that may be used to extend the address space above 4GB [30, p. 58], but since the standard specifies it as optional, only very few OHCI chips support it [29].

Witherden conducts their DMA attacks by utilizing *libforensic1394,* their custom written successor to the software utility from Boileau that also works for more modern OHCI-stack implementations in Linux Kernels and introduces little performance overhead for requests.
Using *libforensic1394,* they were able to successfully exploit (then-) state-of-the-art OS versions, adding to the list set out by previous publications [31]:

- MacOS X 10.6
- Windows XP SP3
- Windows Vista SP1
- Windows 7 (x86/x64)

They also elaborate on active applications of forensic access (writing to memory) such as bypassing the requirement for correct credentials in order to log into a target system.
This is achieved by writing to the memory location that contains the function the OS uses to compare the user entered password against the correct one, negating the comparison operator. For this purpose, they provide the necessary memory signatures for the previously mentioned OS versions.

Tresor-Hunt [32], a novel attack technique also based on DMA and 1394 has utilized the ability to actively interfere with memory content in a live system to subvert the DMA protection mechanism *Tresor* [33]. This mitigation strategy holds key-material and encryption processes in CPU registers and caches so that they are never released to memory.
However, Tresor-Hunt places an attack payload into kernel memory space via DMA and interferes with control flow to have it executed. It then releases corresponding secrets back into RAM so that they are accessible via DMA again. The attack is OS independent, since it relies on data structures required by instruction set architecture (e.g. kernel paging structures and Interrupt Descriptor Tables (IDT)) to determine suitable addresses for payload injection as well as control flow redirection. The authors stress the need for adequate DMA security, emphasizing device whitelisting and IOMMU[10] as possible ways to mitigate.

[10]An I/O Memory Management Unit (IOMMU) is the standard DMA mitigation today. Its functionality and properties are discussed in Section 4.5.

With continuing technological advancements, the introduction- and market penetration of newer interfaces for peripheral devices and I/O has gradually replaced legacy interfaces such as FireWire, PC Card or ExpressCard. This has lead to the emergence of a more diverse set of hardware-platforms for DMA attacks:

- Raspberry Pi [34] and various other USB-connected (development-) boards [35]
- Baseboard Management Controllers (BMC) [36] found on server mainboards
- PCIe extension cards with modified soft- and firmware [37]
- Custom built platforms, often based on programmable FPGA logic [38] [39] [35]
- Living-off-the-land by compromising[11] an already installed device (e.g. network interface card) and subsequently using it to carry out the DMA transfer [40] [41] [42] [43]

In general, any malicious peripheral device that connects using a bus system with direct bus (and therefore memory-) access not mediated by an intermediate protocol between requester and completer can carry out malicious DMA transfers. For this reason, the majority of PCI(e)-connected bus systems are able to carry out such an attack (e.g. 1394, Thunderbolt, M.2, U.2 etc.).

Continuing this train of thought, the process of carrying out a DMA exploit always depends on the ability to perform DMA transactions. Regardless of whether this ability is gained by exploitation of another vulnerability in a device or the introduction of a malicious device to the system, the DMA attack itself always works by reading from- and writing to RAM, using the DMA facilities provided by the underlying hardware platform in a mostly unmodified and intended way, albeit with illicit purpose.
Therefore, further discussion of DMA transactions on a bus-specific level as displayed in Figure 5 and Figure 8 are out of scope for the above-mentioned interfaces as they mostly pertain architecture specifics of the chosen bus such as transport layer logic, request formation, -forwarding and -handling as well as bus arbitration mechanisms.

In a similar fashion, the outcome of successfully establishing DMA connectivity to memory are also expected to be the same across attack implementations[12]. Since full access to memory is granted, any memory location can be read and written. Therefore, anything that resides in RAM can be acquired for further forensic analy-

[11]Initial foothold in these attack scenarios may be achieved by exploiting vulnerabilities in firmware or low-level management utilities of such devices.

[12]With the exception of reampped DMA access with enabled IOMMU

sis (memory imaging) or arbitrarily manipulated by injecting code (e.g. subverting full disk encryption).

In IT-forensic use-cases where the source device may be chosen freely, FPGA-based attack platforms[13] seem to be favorable compared to alternatives because they entail fewer limitations:

- They are able to natively address 64-bit [35], allowing access to high-memory space, significantly contributing to image completeness
- This property also eliminates the requirement for code-injection in fully passive forensic applications in order to access RAM above 4GB, improving reliability, image consistency and tool marks
- Albeit gateware specific, the higher performance offered by FPGA platforms accelerates image acquisition, reducing image inconsistencies
- Due to the *programmable circuit logic* property of FPGA's, they can be designed for the attack purpose in mind, allowing for highly customizable implementations independent of bus interface used

Regardless of attack platform architecture, readily available tooling for software and platform gateware allows for more straightforward implementation and deployment of DMA-based forensic acquisition solutions.

Inception, a software tool released in 2011, provides a centralized command line interface for passive- and active memory forensic applications [44]. This is accomplished by impersonating an IEEE 1394 device against the target system and therefore be granted DMA access to memory. The author also maintained a list of memory signatures for Windows XP, Vista, 7, 8, 8.1, Ubuntu, Linux Mint and MacOS X.

Inception has been superseded by *PCILeech* [35], a similarly versatile framework for forensic memory acquisition. U. Frisk released PCILeech alongside their 2016 publication featuring a DMA attack against then state-of-the-art Windows, Linux and MacOS OS versions [37] conducted from a USB3380 development board.
Some distinguishing features from PCILeech include platform support for various hardware platforms as well as software acquisition methods, options for a graphical

[13]A Field Programmable Gate Array (FPGA) chip as opposed to a CPU is a computing chip that can be freely customized by programmable circuitry and does not need to adhere to a specific instruction set or internal architecture specification

user interface in addition to CLI, readily maintained memory signatures for different operating systems as well as resources and tools on creating custom signatures and code to inject, a comprehensive feature set for active applications and high data transfer speeds of up to 150MB/s.

As of 2023, PCILeech has since been maintained and gained a substantial amount of traction in the academic community focussing on memory forensics. During our previous work, we determined whether a PCILeech/FPGA attack platform was able to acquire memory from a sample of operating systems whilst adhering to a realistic threat model for forensic investigators.

PCILeech has also been used for more complex forensic use-case implementations such as *BMCLeech* [36]. Baseboard Management Controllers (BMC) are PCIe-connected coprocessors on enterprise mainboards that allow platform management and monitoring capabilities independent of CPU architecture, firmware and OS [45]. BMCLeech utilizes this aspect to provide on-demand memory forensics for servers that is very stealthy and leaves significantly little inconsistencies and tool-marks on acquired images, as no new device needs to be introduced into the system.

Deployments of BMCLeech require prior setup in a system (rollout of BMCLeech onto the BMC), but allow for on-demand memory forensics in a remote fashion. The proposed use-case enables forensic investigators to run a PCILeech instance on a remote analysis workstation that connects via network to the BMC of the target machine.
BMCLeech itself runs on the coprocessor and is built upon *OpenBMC*, an open source BMC firmware stack. The actual DMA access to RAM is conducted by a customized kernel driver. Because of this, the target host is incapable of detecting whether the BMC is currently carrying out malicious DMA accesses apart from determining that a BMC is present on the system.
The authors have evaluated memory images from BMCLeech against software-acquired images by *LiME* [46] and found that their implementation produced images of equal consistency and forensic soundness than LiME.

Albeit limited by the requirement for an OpenBMC compatible chip and disabled IOMMU protection, BMCLeech presents a novel and promising usage scenario for on-demand memory image acquisition, especially for blue teams and incident response departments.

HyperLeech, a novel approach to DMA based memory forensics presented by Palutke et al. uses the PCILeech framework to inject a lightweight hypervisor into the target memory, transitioning the OS from regular execution to a VM inside the hypervisor [47]. This allows circumvention of sophisticated anti-forensic measures employed by advanced malware and enables forensic analysts to interact with the target system in a live-state similar to a VM. During their insightful research, the authors demonstrate a multistep process for code injection and the transitioning process of the "physical OS" to the virtualized state as well as how to revert these changes.

They further emphasize the need for IOMMU protection that properly mitigates the DMA attack vector, as HyperLeech may also be used maliciously as a low level rootkit. As their demonstration limits itself to Linux operating systems and Intel CPU architectures, a corresponding adaption for AMD and ARM architectures as well as Windows- and MacOS might be a promising lead for future iterations of HyperLeech.

Concluding this section about the different means of DMA exploitation, we would like to mention that findings of general hardware- and firmware vulnerabilities heavily contribute towards the relevancy of DMA as an acquisition method, since there always needs to be a way to gain illicit access to memory via DMA and emerging mitigation measures make it increasingly difficult to do so.

For example, the novel findings about the widespread Thunderbolt protocol by B. Ruytenberg [48] revealed several vulnerabilities that practically break Thunderbolt 3 security, in turn allowing DMA access to memory. Other creative approaches for unauthorized memory access include impersonation of other, lesser known endpoints on the bus, such as introducing a rogue memory controller into the system [49] or abusing debugging capabilities brought into silicon by CPU Vendors (e.g. Intel Direct Connect Interface [50]) for forensic memory access.

Mitigation Implementation: Windows

Now that the domain of DMA attacks has been covered extensively, the relevant means of protection against them should be discussed, starting with Microsoft's operating systems.

The security feature developed by Microsoft is called *Kernel DMA Protection* and is henceforth abbreviated throughout this thesis as KDP to improve readability. KDP is available for Windows 10 and 11 operating systems [51].
Although marketed as *Boot DMA Protection* under the term *Secured-Core Server* alongside other security features, the same KDP technology also applies to Windows Server systems [52].

KDP does not implement its own prevention mechanism, but rather relies on one of the most common security measures brought in place against attacks relying on malicious peripheral devices, an *I/O Memory Management Unit* (IOMMU). The in-depth function of an IOMMU is discussed in Section 4.5, but generally speaking, it allows the system to perform *memory isolation* [51] for peripheral devices so that they are only able to access memory regions that have been assigned to them. This remapping works similarly to how I/O ports are mapped to memory locations as has been explained in Section 2.2.2.

Depending on whether a given device driver supports DMA remapping, KDP will allow the OS to enumerate and start remapping capable devices immediately[14].
Devices with drivers incapable of remapping will not be allowed to start if they were connected to the system before user authentication took place or during non-user-authenticated (i.e. locked-) system states (S1-S5) [53].

Once the system has been unlocked by an authorized user, the device driver is started by the OS and the peripheral will start its normal function (until the next reboot) and continue to do so even if the system is locked or the user signs out.

[14]PCIe device enumeration means discovering the presence of- and granting a logical PCIe link to the device on the bus, allowing it to perform transactions

Pamnani and Matarazzo [51] mention that KDP is supported on the following Windows 10/11 editions:

- Windows Education (Licenses A5/A3)
- Windows Pro / Windows Pro Education/SE
- Windows Enterprise (Licenses E3/E5)

Most notably, the Windows Home edition does not seem to be included. Additionally, Microsoft provides an extensive list of platform requirements [54] for OEMs pertaining KDP.

Having reviewed KDP and its core functions, the authors would like to acknowledge some of the shortcomings and weaknesses that it entails.
Firstly, KDP does not protect against DMA attacks via some legacy interfaces such as 1394/FireWire, PCMCIA, CardBus and ExpressCard [51]. It also is not compatible with other BitLocker specific DMA countermeasures, in fact it is recommended by Microsoft to prioritize KDP.

Furthermore, KDP does not provide any protection against DMA exploits during the boot process [51] with documentation stating firmware- or BIOS level protections being required for this specific attack vector.
Finally, the authors previous work [6] demonstrated that KDP is not turned on by default on all Windows operating systems.

The authors would also like to mention two closely related technologies adjacent to KDP as discussed here: *Windows Kernel Data Protection* and *Virtualization-based Security* (VBS) [55]. The latter makes the Windows kernel run in specially secured virtualized environments to protect the OS kernel code itself and provide a root of trust for the OS, introducing a threat model where the kernel can be assumed compromised.

Windows Kernel Data Protection [56] should also be mentioned shortly. It intends to protect the kernel against data-driven attacks via two parts:

- *Static:* Kernel mode software can now statically protect a section of its own image from tampering by any other entity.
- *Dynamic:* Kernel mode software can allocate and free memory from a secure pool. Memory from this pool is read-only and may only be initialized once.

Mitigation Implementation: MacOS

In typical Apple fashion, the provided material documenting their means of OS-side IOMMU implementation is rather scarce. For Mac computers, a distinction is made by Apple depending on whether it is based on Intel- or Apple silicon[15] [57]:

Apple Silicon

As stated by the manufacturer, Apples SoC's (System on Chip) contain a separate IOMMU for every DMA agent in the system - This enables them to not require DMA interrupt remapping. Apart from that, they shortly elaborate that connected peripherals may only interact with memory that has explicitly been mapped for their usage, as is the normal functionality for IOMMU memory mapping.
Additionally, the Apple-Support resource [57] reveals that if a peripheral should try to access a memory address outside its mapped area, a kernel panic is triggered.
[16]

Intel Silicon

For Macs with Intel processors, Apple states vaguely that the IOMMU based DMA protection mechanism relies on Intel VT-d[17] [59] and that DMA address space- as well as -interrupt remapping [57] happens "*very early in the boot process*" to mitigate corresponding vulnerabilities. Also, a default-deny policy is applied so that DMA requests are blocked, unless explicitly permitted.

Finally, Apple states that the hardware and software changes introduced from macOS 11 onwards facilitate additional security by mitigating malicious device-to-UEFI-driver interactions at boot time [57], especially regarding how drivers handle DMA data buffers.

[15]At some point, Apple switched from Intel based processors for some of their products to in-house designed silicon. Therefore, older Mac models may contain Intel processors while newer iterations rely on Apple chips.

[16]A kernel panic is when the OS encounters a fatal problem or error so that system integrity can no longer be guaranteed, and it is not desirable to continue operating the OS, but rather have it crash [58]

[17]Intel Virtualization Technology for Directed I/O

Mitigation Implementation: Linux / Unix

When it comes to Linux or Unix operating systems, more in-depth documentation in regard to their inner workings, especially the Linux kernel are available due to their open-source nature. For this reason, this chapter is going to be more comprehensive on how these OS' implement I/O Memory Management Units.

Generally speaking, an IOMMU is closely related to a regular Memory Management Unit (MMU). An MMU translates virtual addresses visible to the CPU into physical addresses in memory whereas an IOMMU translates virtual addresses visible to devices into their physical counterpart. These units are hardware components requiring corresponding firmware as well as a software implementation in the operating system so that it can remap address spaces for devices. This part of the kernel is referred to as the IOMMU subsystem [60] and is separated into multiple layers:

- *IOMMU DMA Layer:* Receives and forwards DMA requests from device I/O controllers to the IOMMU generic layer, therefore acting as a translation layer between the respective DMA- and IOMMU-API's
- *IOMMU Generic Layer:* Provides generic API's to the hardware specific IOMMU layer.
- *Hardware Specific IOMMU Layer:* Consists of hardware proprietary drivers that interface the generic API's with the underlying IOMMU hardware. It is also responsible for correct configuration of the I/O page table.

The handling of DMA requests in the Linux kernel depends on various interactions between the device drivers on either side of the transaction (CPU and I/O controller of the device), the DMA subsystem and the IOMMU subsystem, as illustrated by Figure 9.

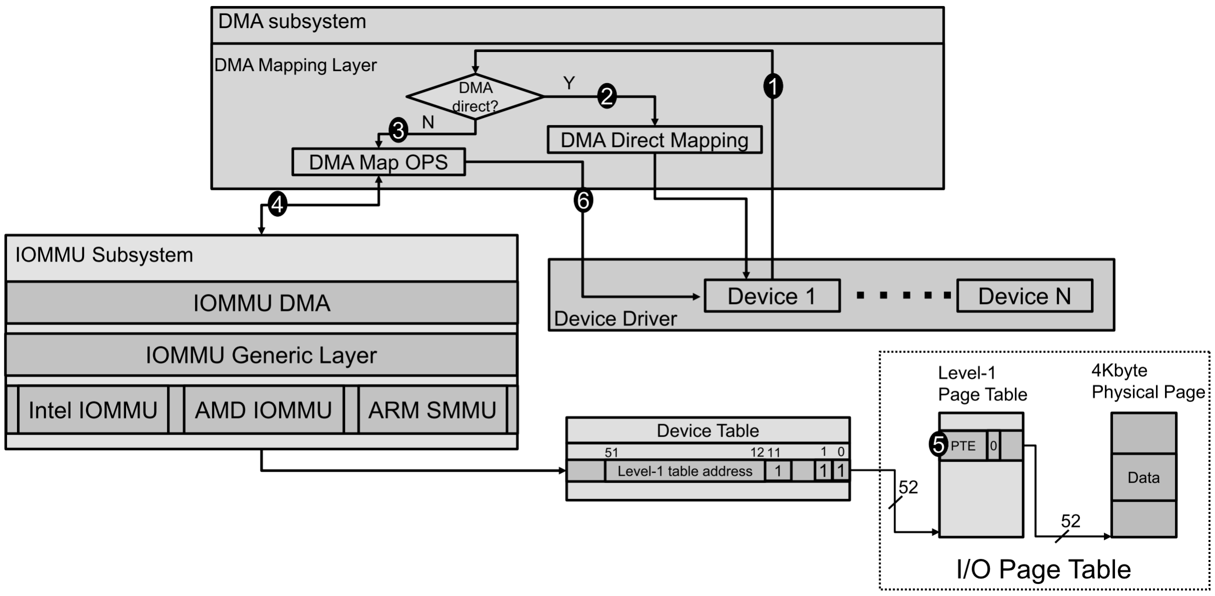

Figure 9: A control flow of DMA Request-Handling on Linux, Source: [60]

A DMA request that has been initiated by a device is handled via the following process in Linux operating systems [60]:

1. Device driver calls API methods to retrieve desired physical memory address.
2. If no DMA remapping takes place (direct mapping - no IOMMU protection), the DMA subsystem returns the physical address directly to the device.
3. Otherwise, the DMA requests are passed onto the IOMMU subsystem (DMA remapping)
4. Previously discussed layers of the IOMMU subsystem translate received requests via API calls
5. IOMMU subsystem performs memory mapping - The hardware specific driver configures the I/O page table so that the actual IOMMU hardware can correctly translate addresses.
6. DMA subsystem returns translated physical address to the device driver

It should be mentioned that Figure 9 depicts a single-level address translation mechanism. IOMMU hardware and their OS implementations also support two-stage- [60] [61, p. 40] [62, p. 40] or nested translation [63, p. 19], which is required for virtualization applications (mapping to either the physical address space of the host- or guest OS).

I/O Memory Management Units

IOMMU protection is a mitigation strategy against DMA exploits that encompasses multiple layers of the hard- and software stack such as the OS, UEFI/BIOS, Firmware/Drivers and even down to the hardware level in the form of IOMMU hardware that is an integrated part of the chip silicon in modern processors.

Due to this circumstance, IOMMU hardware implementation differs depending on the CPU manufacturer and hence needs to be supported by specific drivers and API's in the software and firmware layers built on top of it [64, p. 16]. One such case would be the previously discussed layers of the IOMMU subsystem in the Linux kernel.

The OS needs to be notified about the IOMMU capabilities and -properties of the underlying hardware platform [60]. The according information is provided to the OS via an ACPI[18] table containing data structures that describe I/O virtualization.

Extensive system architecture documentation is openly available regarding the IOMMU implementations of the three main processor architectures:

- Intel Virtualization Technology for Directed I/O (VT-d) [61]
- AMD I/O Virtualization Technology (IOMMU) [62]
- ARM System Memory Management Unit (SMMU) [63]

The remainder of this section is going to discuss these (hardware) IOMMU implementations on a high level, examining their main capabilities and providing a basic understanding of how they try to achieve device remapping, especially in regard to how they differentiate between different devices, also shortly covering architecture topologies where applicable. In-depth discussion of address translation algorithms, data structures or detailed operating procedures are out of scope.

[18]ACPI is an open standard originally made by Intel, Microsoft and Toshiba [65] that operating systems can employ for hardware component discovery, power management, auto-configuration and status monitoring.

Intel VT-d

It should be mentioned that IOMMU usage has not been explicitly designed as a security mitigation measure against DMA attacks, but rather as a virtualization technology for I/O. When the requirement for a DMA attack mitigation arose, manufacturers adapted so that their remapping techniques for I/O memory address space virtualization and -isolation which was originally meant to separate virtualized hosts and restrict them to their respective memory address spaces could also be used for this use case. Therefore, implementations may provide features that may not be relevant in the context of DMA exploitation. Intel describes the capabilities of their VT-d technology as follows [61, p. 21]:

- *I/O Device Assignment:* Enables flexible assignment of devices to virtual machines and extending the isolation of virtualized hosts to include and protect I/O operations
- *Interrupt remapping:* Isolation and routing for interrupts from devices or external interrupt controllers to the correct virtual machines
- *Interrupt posting:* Allows virtual interrupts from devices or external interrupt controllers to be directly forwarded to their respective virtual CPUs.
- *Reliability:* Capabilities to record and report errors in regard to DMA or interrupts that could corrupt memory or otherwise negatively impact the isolation of virtualized guests.
- *DMA Remapping:* Address translations for DMA from devices. Apart from security mitigations, it also allows operating systems to allocate non-contiguous memory to devices [66] (remapping as an abstraction layer makes it seem contiguous for the device I/O controller) and enables support for legacy 32-bit devices to access high-memory regions in 64-bit systems.

Intel achieves DMA remapping via the introduction of *domains* [61, p. 22], an abstract (and therefore OS- and architecture independent) isolated environment that has had a subset of the host memory allocated to it.

The isolation of device access to system memory is enabled by assigning devices to domains, with each DMA-capable device being assigned to at least one domain [64, p. 17]. Additionally, the OS is provided the capability of flexibly changing domain assignments for a given device. The domain assignment mechanism is implemented via a distinct set of paging structures and the respective device-domain mappings (contexts) are stored in a context cache that is part of the DMA remapping hardware (DRH).

The DRH is the part of the IOMMU that is responsible for address translation and intercepts every DMA request to memory and decides based on the page tables whether the access may be permitted [61, p. 22]. This is based on whether the requested address is part of the memory address space assigned to the domain the requesting endpoint belongs to. If not, the DMA request is blocked. Since the overhead of such processes needs to be kept to a minimum as to not impair system performance, a lot of the required mechanisms are implemented in silicon and high-performance hardware caches are used for storing and quickly accessing frequently used data structures such as the involved paging structures.

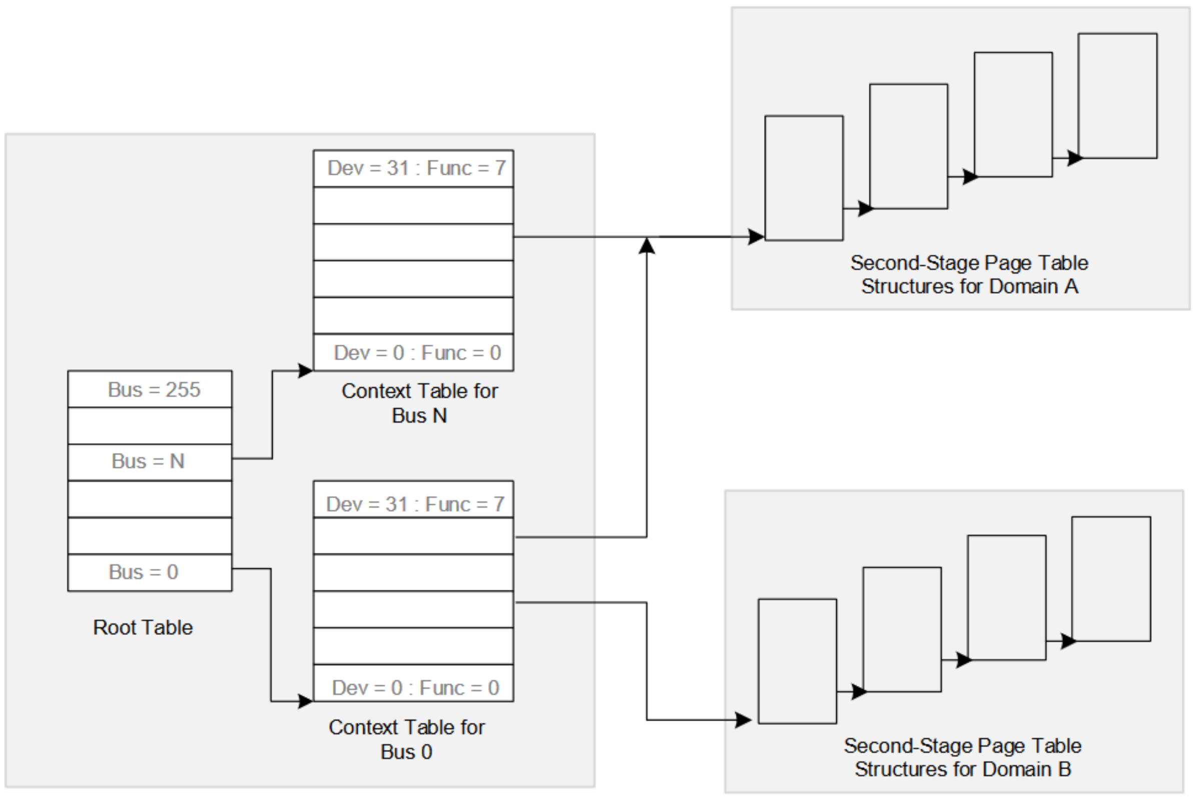

Figure 10: Levelled (Multi-Stage) Translation Tables in Intel VT-d, Source: [61]

As depicted in the hierarchical structure in Figure 10, the root table is the top level structure to map devices to their respective domains [61, p. 31]. It is 4KByte large and consists of 256 root entries to cover the entirety of the PCIe bus number space. The root table entries reference entries in context tables, which map a specific I/

O device to its respective domain and the address translation data structure necessary for it. The 256 entries in the context tables in turn correspond to a PCIe device function on the bus. In addition to domain isolation, DMA remapping is also utilized to protect the operating system itself as well as other important memory regions via *Protected Memory Regions* (PMR) and the *DMA Protected Range* (DPR) functionality [67].

PMR's are two memory regions that are protected against DMA access [67]:

- *Protected Low Memory Region:* Must be in the lower 4GB of memory and holds hypervisor code and data as well as the initial DMA-remapping structures for address spaces below 4GB
- *Protected High Memory Region:* Must be above 4GB and may be variably sized, but has to be big enough for the initial DMA-remapping structures for address spaces above 4GB

A DPR is a contiguous region of system memory that is protected from all DMA accesses [68], even if they passed VT-d translation. DMA transactions that pass and hit this address range are faulted. The size of this memory segment is set and locked by BIOS. This way [67], a memory range can be provided that is only available to processor streams.

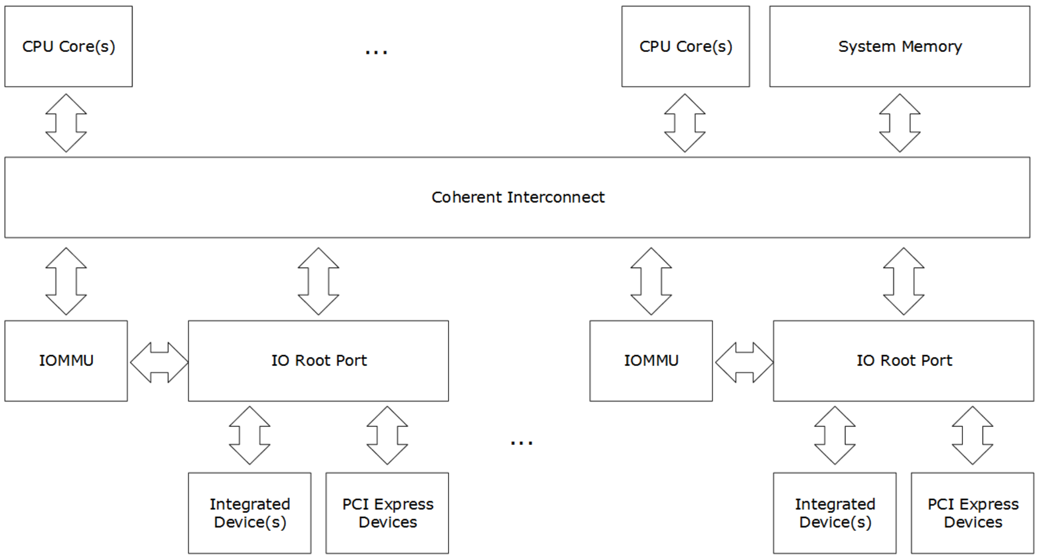

Figure 11: Intel VT-d Platform Topology, Source: [61]

Figure 11 illustrates a high-level topology overview of an Intel platform and where an IOMMU is positioned in the architecture of intel silicon.

AMD IOMMU

Regarding its capabilities, AMD's IOMMU counterpart exhibits an architecture (Figure 12) as well as a feature-set very similar to Intel [62, p. 31]:

- Direct access to user space I/O for devices
- Direct I/O access to devices by guest VM's
- Isolate Devices to prevent malicious DMA access to system memory
- Legacy I/O Support for 32-bit devices on 64-bit operating systems
- Improved Security for I/O device access from user applications and VM's
- Interrupt remapping, filtering as well as direct interrupt delivery to VM's

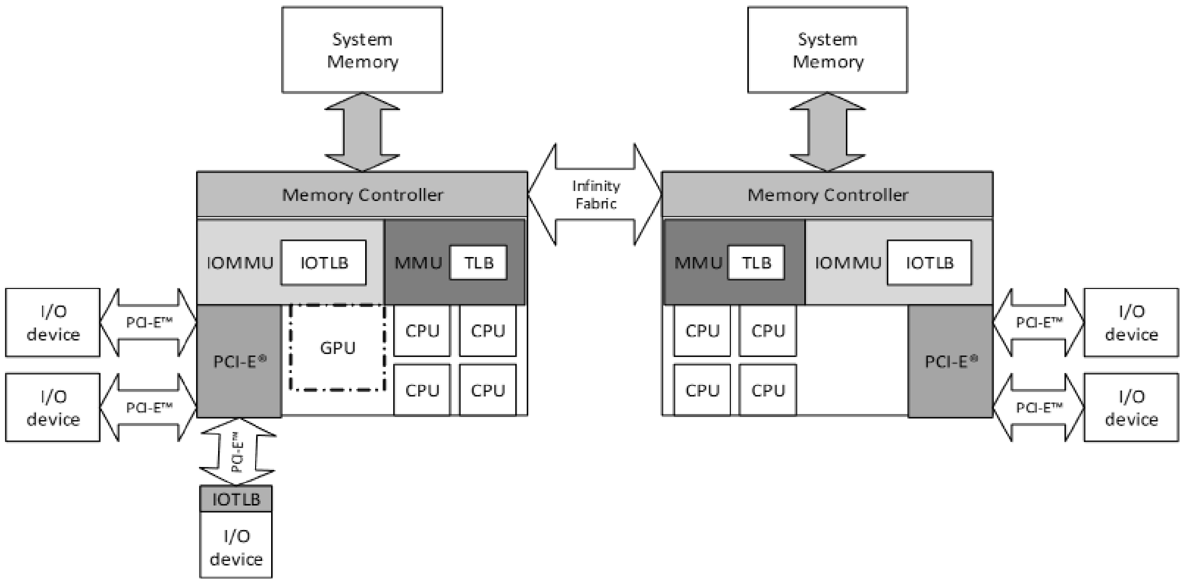

Figure 12: Example of an AMD IOMMU Platform Topology, Source: [62]

AMD have had a precursor technology called *AMD64 DEV* prior to their introduction of IOMMU that already performed some sort of protection domain assignment to devices which was then built upon [62, p. 31]. Their IOMMU implementation assigns each device to a specific domain and a distinct set of page tables.

When an I/O controller of a device performs read or write actions on memory, the corresponding transactions are intercepted by IOMMU. It then determines the domain-assignment of the calling device and uses Translation Lookaside Buffers (TLB's) belonging to that domain and I/O page tables belonging to the device to

determine whether the access is to be permitted as well as the actual location in memory.
Analogue to Intel, the AMD utilizes an *IOMMU Device Table* that associates I/O devices to their domains and also contain pointers to the I/O device's page tables [62, p. 32]. An example Entry for this table is given in Figure 13:

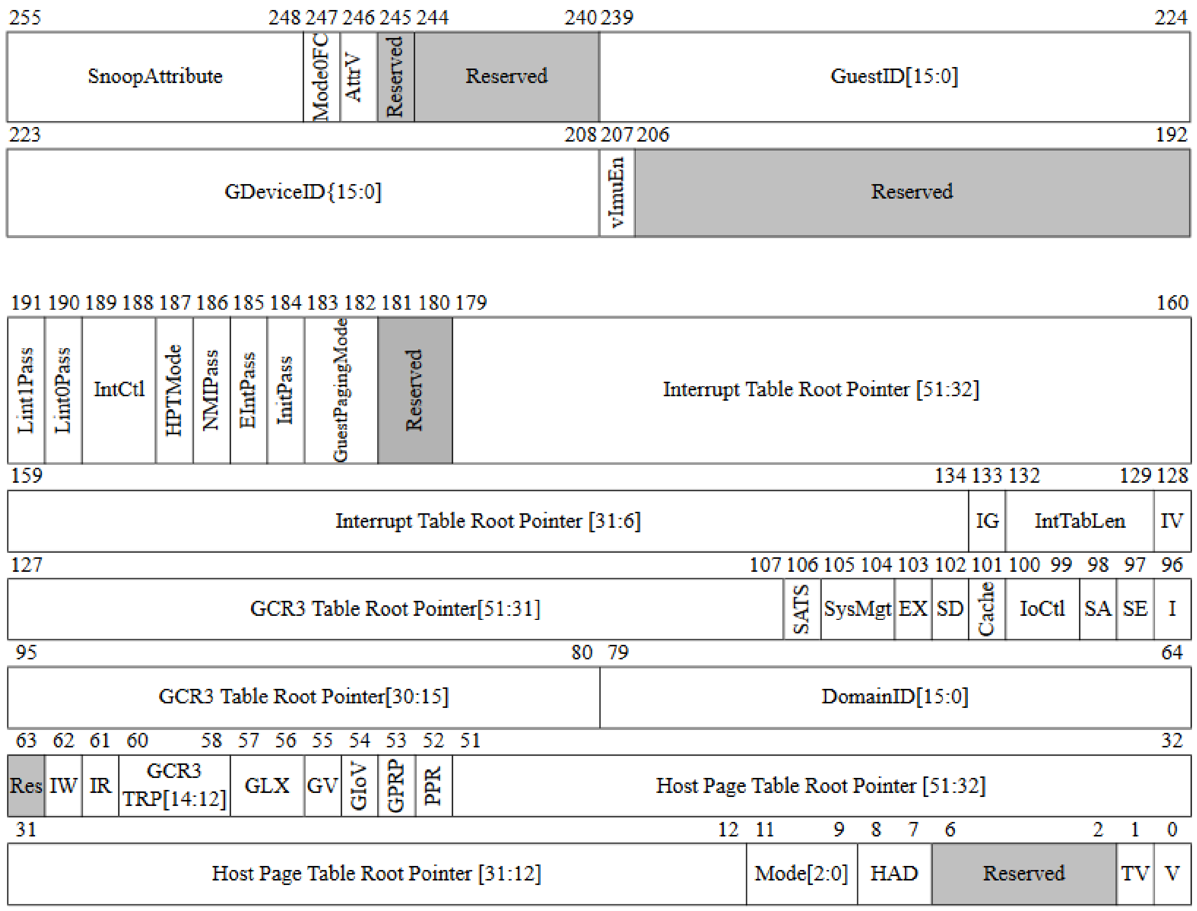

Figure 13: AMD IOMMU Device Table Entry, Source: [62]

The entry contains pointers to several page- and interrupt-tables, control- and status bits (e.g. controlling Interrupts via `IntCtl`) as well as a unique identifier for the device the entry corresponds to, as well as a `DomainID` for domain assignments [62, p. 63]. As indicated by the depicted address range from the entry, the AMD device table entries are 256 bit in size, whereas Intel root table entries are 128 bit[19].

[19]With root table size and entry count as per Intel specification [61]: $\frac{4\text{KByte} \cdot 8 \cdot 1024}{256 \text{ Entries}} = 128$ Bit

ARM SMMU

The ARM architecture describes its version of an IOMMU, called an SMMU with a very similar feature-set as the previously discussed technologies [69] [63], hence it is not going to be reiterated. This is due to the fact that the provided capability set is driven by the industry standard demands for virtualization and its core properties of isolation, protection and abstraction. An example of possible SMMU implementations is given in Figure 14.

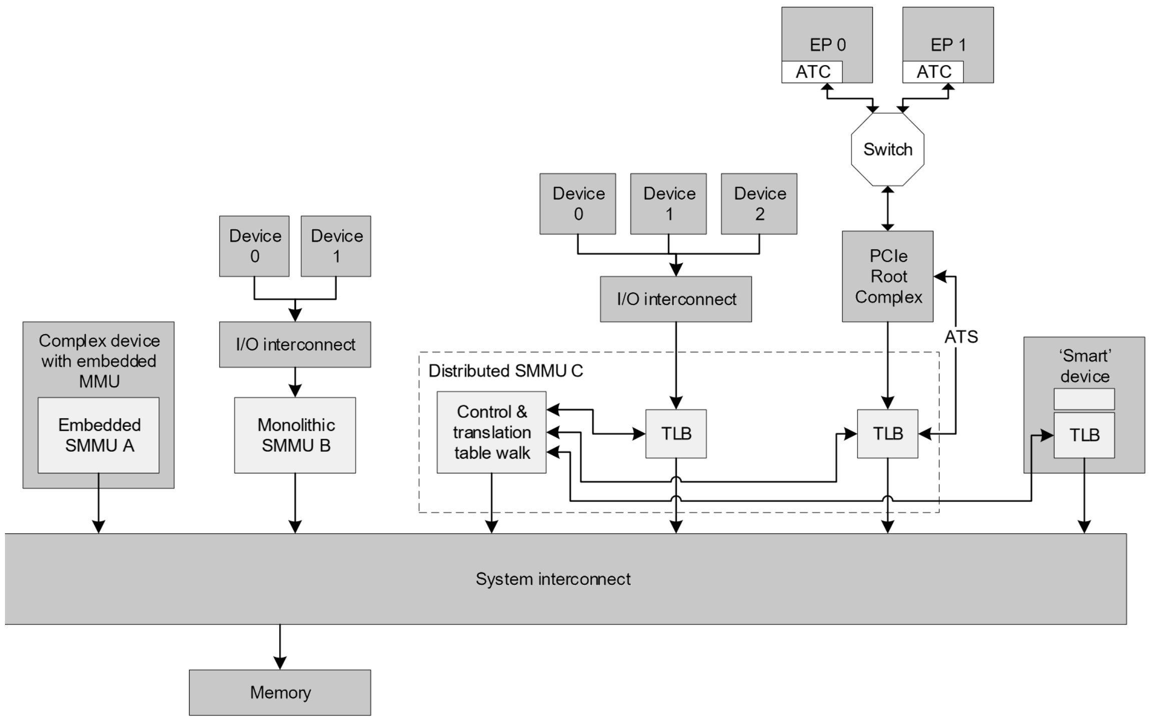

Figure 14: ARM SMMU Topology Examples, Source: [63]

The `Distributed SMMU C` implementation is what can be assumed a pendant to the previously discussed IOMMU topology illustrations for Intel (Figure 11) and AMD (Figure 12) respectively. ARM refers to Requesters (of DMA transactions) as *client devices* of an SMMU [63, p. 29]. The PCIe endpoints EP0 and EP1 would therefore be clients to the SMMU C.

SMMU achieves association between devices and their traffic by introducing stream identifiers - different *StreamID's* therefore logically distinguish different SMMU clients [63, p. 19] from one another.

SMMU also supports nested- or dual-stage-translation in order to satisfy corresponding requirements for virtualization of guest operating systems [63, p. 19]:

- Stage 1 translation is used for regular DMA remapping and translates virtual addresses to *Intermediate Physical Addresses* (IPA)
- Stage 2 translation is used for virtualizing device DMA to guest VM address spaces and translates IPA's to physical addresses.

The multi-stage tables introduced to achieve the described translation scheme share similarities with Intel's root table structure and allows the SMMU to locate the data structures in memory required for translation. The root of this structure is represented by the *Stream Table* [63, p. 36], it can be found via the base address that is stored in registers and is depicted in Figure 15.

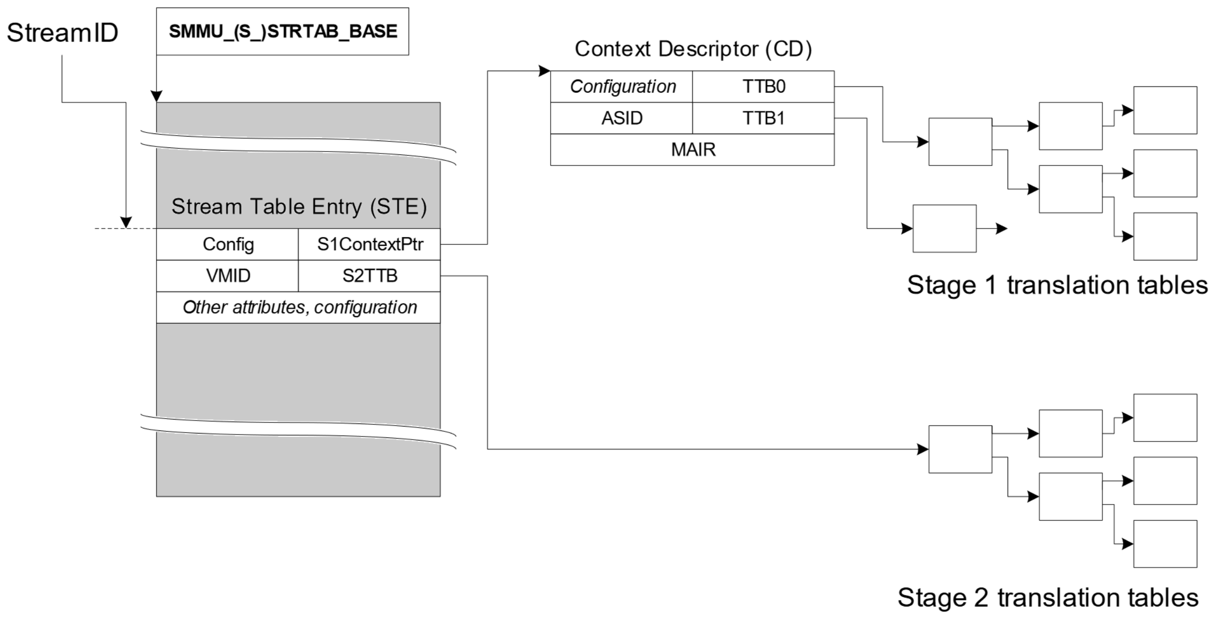

Figure 15: ARM Stream Table Structure, Source: [63]

An entry to this table (STE) contains base pointers to stage 2 translation tables and pointers to stage 1 configuration structures, among other member attributes. STE's represent stage 2 translation, while a *Context Descriptor* (CD) represents stage 1 translation.

Two types data structures used by SMMU's are distinguished [63, p. 36]:

- *Configuration Structures:* Associate a StreamID with its respective translation table base pointers and configuration as well as context.
- *Translation Table Structures:* The actual data structures used to perform address translation between virtual-, IPA- and physical addresses

The translation procedure for intercepted transactions starts with locating the STE according to the StreamID of the request. The STE content informs the SMMU about the additionally required configuration [63, p. 36]:

- Is traffic from this client permitted?
- Does it require stage 1 translation?
- Does it require stage 2 translation?

If stage 1 is used [63, p. 38], the corresponding STE attribute points to the location of the CD in memory. This context descriptor associates the StreamID with stage 1 translation tables.
However, if the transaction is subject to stage 2 translation, the STE references the according translation table base pointers directly, as opposed to via an intermediate data structure such as a CD.

A structural overview of a stream table entry is given in Figure 16. Continuing the previous comparison, it is twice the size of an AMD device table entry with 512 bit [63, p. 179]. Similarly, it reserves a significant amount of bits (as indicated by `RES0`). This is usually done for forward compatibility with future SMMU iterations. The previously mentioned references to stage 1 CD's are stored in the `S1ContextPtr` fields, stage 2 translation table base pointers via `S2TTB` etc.

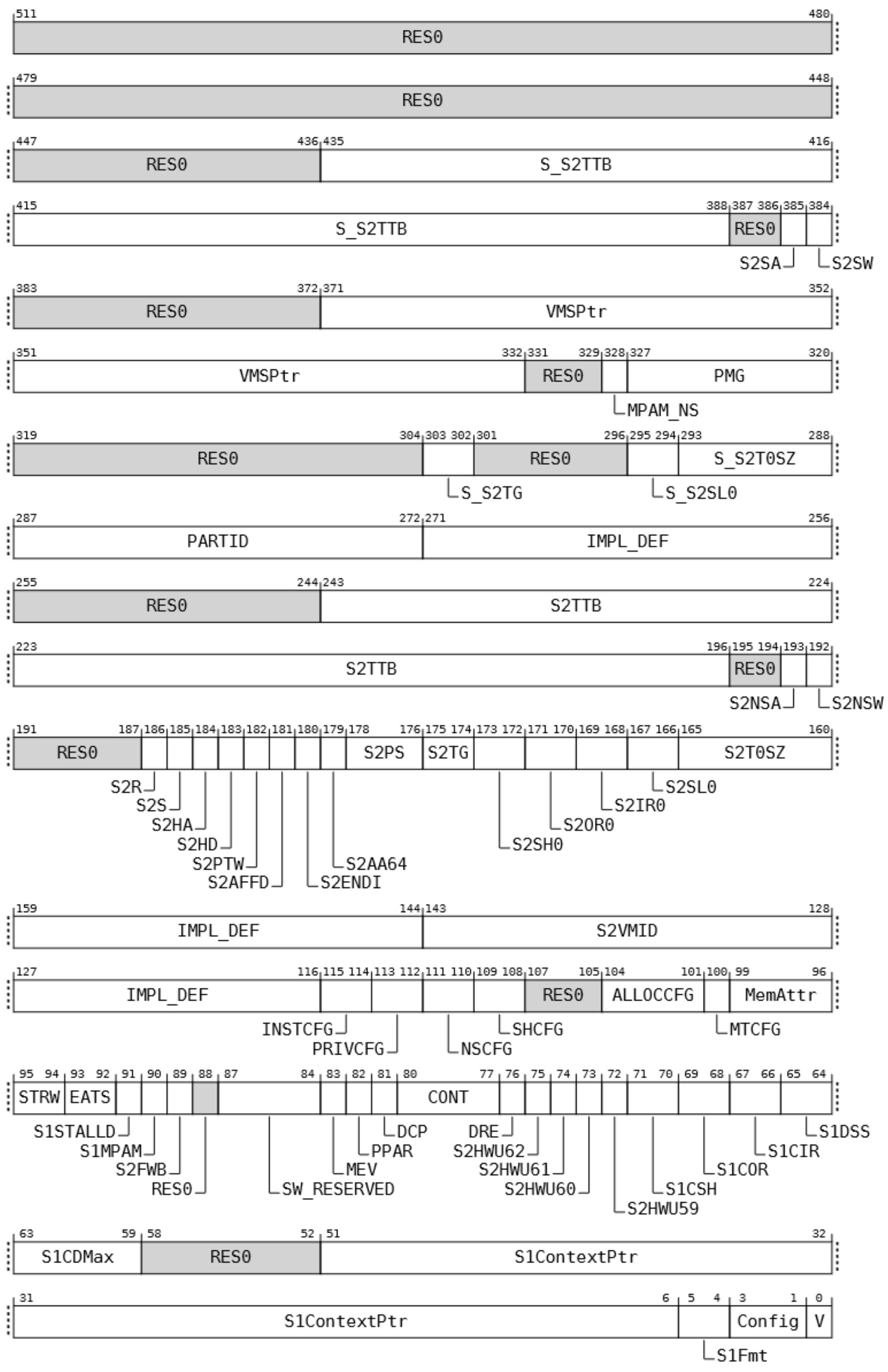

Figure 16: ARM Stream Table Entry Fields, Source: [63]

This concludes the overview of hardware IOMMU units per manufacturer. It should be emphasized that in order to develop circumvention techniques for these implementations, they would need to be investigated on an even more in-depth level, such as trying to reverse engineer mapping algorithms, buffers and caches as has been demonstrated by the successful implementation of DMA attacks despite enabled IOMMU by Peglow et al. [64] and Markettos et al. [39].

The authors highlight that many IOMMU implementations are faulty, often not enabled in OS- and firmware settings and that DMA remapping by itself does not sufficiently protect. They identified several vulnerability families applicable to systems with enabled IOMMU's [39] [70]:

- *PCIe ATS:* Address Translation Services as per PCIe specification allows for devices to implement their own TLB's (e.g. Figure 14 - `SMMU A` and Figure 12 - Device with IOTLB). A malicious device may abuse this by announcing itself ATS-capable and advertising its DMA transactions as "already translated" so that they aren't intercepted by an IOMMU. This allows for full IOMMU bypassing.
- *Kernel Pointer Exposure:* Incorrect implementation or usage of memory buffers may lead to the exposure of kernel pointers, allowing to interfere with control flow of kernel traffic.
- *Spatio-Temporal Vulnerabilities:* Tampering with device data structures such as registers and ring-buffers mapped to memory allows data in different memory locations to be accessed
- *Shared Allocators:* Implementation errors on driver- or kernel-side memory allocation may expose sensitive memory content to peripheral devices.
- *Sub-Page Vulnerabilities:* Whenever an I/O location is DMA remapped, all information from the physical memory pages it is mapped to become accessible to the I/O device because IOMMUs work with per-page granularity. Thus, if the mapped location is smaller than a memory page, potentially sensitive information from the rest of the page can be accessed by the I/O device.

Further in-depth discussion of IOMMU specific vulnerabilities is out of scope for this review, as they are deeply involved with kernel memory management, -allocation and -control-flow as well as low level manipulation of data stores, -structures or drivers, hence being vast knowledge domains warranting a separate review.

Conclusion

During this work, we synthesized the knowledge body of *Direct Memory Access attacks* and established mitigation techniques against it, especially focussing on applications of such exploits for forensic acquisition of system memory. Following this synthesis, a taxonomy of the examined literature is provided in Section 8.

The various bus systems and -interfaces for connection of peripheral devices can be divided into *Expansion-* and *Peripheral-Busses*. The former have direct bus access and can therefore be used for DMA access to system memory. They include PCIe, FireWire, Thunderbolt and M.2 among others. During our survey, we covered PCIe and FireWire in detail. The successful execution of DMA attacks only partly relies on the architecture- and transport layer specifics of the underlying bus system, as the DMA facilities are used mostly in an intended way, however with illicit purpose (accessing memory locations outside of what a device should be able to interact with).

This property is what makes DMA an attractive memory acquisition method. Although several publications have shown ways to exploit firmware- and hardware vulnerabilities for initial access to a peripheral device already installed in order to then execute the actual DMA transaction on a target system, the majority of IT-Forensic application scenarios are likely going to rely on the introduction of a malicious device into the target.

This may either be done preemptively to allow for on-demand memory forensics or ad hoc, satisfying the requirement for target system access in threat models where the forensic investigator has no legitimate way of accessing locked systems. Upon successful application, DMA acquisition may even allow circumvention of full-disk encryption and grants arbitrary read- and write access to RAM.
Apart from bus-system and soft-/gateware specifics, a DMA acquisition platform is subject to certain limitations depending on the hardware platform used for the malicious acquisition device. Modern approaches seem to favor FPGA-based devices, as they can natively address 64-Bit, are highly customizable and surpass their alternatives in data transfer performance.

Following our survey of DMA usage for memory imaging, we conclude that DMA based methods are able to produce RAM images of significant forensic soundness:

Prerequisites:	As exhibited by our previous work, DMA attack platforms may be built inexpensively using off-the-shelf hardware and open-source software.
Completeness:	Full memory access and the ability to place and execute code allow for forensic acquisition of locations outside of system memory such as CPU registers, caches, buffers etc.
Consistency:	Imaging performance is bus-, protocol- and acquisition-platform specific. In case of PCIe/FPGA based solutions, they can reach very high transfer speeds (>150 MByte/s).
Reliability:	As DMA is a hardware acquisition method, it introduces very little changes into the system state apart from acquisition device introduction as no code needs to be executed for imaging.

The asserted limitations mostly pertain the feasability of DMA acquisition on a given system as opposed to image quality metrics:

Tool Marks:	Introduction of the acquisition device to a host while running usually leads to a re-enumeration of devices on the bus as per ACPI specification, resulting in changes to the current system state.
Risk Factor:	Depending on bus used, connecting the acquisition platform during a running system state may lead to undocumented behaviour, especially for non hot-plug capable interfaces such as PCIe, potentially crashing the system.
Mitigations:	Enabled IOMMU protection may prevent DMA access to full-memory and therefore protect against common DMA attack frameworks. However, the reviewed literature suggests a very broad vulnerability space related to IOMMU's and multiple researchers have shown DMA exploits despite enabled IOMMU protection.

Future Work

Finally, we have identified several areas for future approaches to DMA in forensic applications, which are going to be discussed in the following section.

DMA Forensics in non-IT Environments
The heterogeneous distribution of bus- as well as interface technologies across different industry branches leads us to believe that there is value in pursuing DMA forensics in OT environments specifically. OT systems have fundamentally different requirements, architecture- and design specifications they need to adhere to. Ultimately, there are many bus-systems employed in today's operational technology sectors that have not been examined in the context of DMA acquisition feasibility. This is further emphasized by the fact that OT systems often have significantly longer lifespans and are likely to still utilize technologies otherwise considered legacy today.

Alternate Technologies as Access Vectors
Similarly, the exploration of new initial access vectors for DMA attacks may also contribute towards the DMA knowledge domain. Apart from reverse engineering efforts and attacks aiming to find new vulnerabilities for hard- and firmware of peripheral devices, we would like to suggest researching bus-/interface technologies that have not been examined in the context of DMA forensics so far.

DMA & IOMMU Vulnerabilities by CPU Vendor
A large portion of the examined literature on DMA exploitation performed their attacks in Intel-based systems. This is likely due to the fact that until recent years, Intel had a significantly larger amount of market share when compared to AMD and ARM. Because of that, research on DMA and vulnerabilities in the IOMMU implementations of these CPU families may lead to promising results, especially since they employ analogues for Intel technologies known to be vulnerable to DMA attacks (e.g. ARM/AMD counterpart for Intel DCI).

Overcoming the IOMMU Hurdle
Finally, one of the most important contributions to DMA-based forensics according to our survey is the integration of IOMMU circumvention techniques into commonly used DMA frameworks and hardware. This would allow DMA to become a reliable memory acquisition method, even when faced with IOMMU protection.

Acronyms

DMA	Direct Memory Access
I/O	Input / Output
VM	Virtual Machine
FPGA	Field Programmable Gate Array
PCI(e)	Peripheral Component Interconnect (Express)
ISA	Industry Standard Architecture
PCB	Printed Circuit Board
OS	Operating System
MMIO	Memory Mapped I/O
FSB	Front Side Bus
RC	Root Complex
TLP	Transport Layer Packet
OHCI	Open Host Controller Interface
PCMCIA	Personal Computer Memory Card International Association
FDE	Full Disk Encryption
BMC	Baseboard Management Controller
NIC	Network Interface Card
DCI	(Intel) Direct Connect Interface
API	Application Programming Interface
KDP	(Microsoft Windows-) Kernel DMA Protection

VBS	(Microsoft Windows-) Virtualization-based Security
MMU	Memory Management Unit
IOMMU	I/O Memory Management Unit
OEM	Original Equipment Manufacturer
BIOS	Basic Input Output System
SoC	System on Chip
VT-d	(Intel) Virtualization Technology for Directed I/O
UEFI	Unified Extensible Firmware Interface
ACPI	Advanced Configuration and Power Interface
SMMU	(ARM) System Memory Management Unit
DRH	DMA Remapping Hardware
PMR	Protected Memory Region
DPR	DMA Protected Range
TXT	(Intel) Trusted Execution Technology
TPR	(Intel) TXT DMA Protection Ranges
TLB	Translation Lookaside Buffer
IPA	(ARM) Intermediate Physical Address
STE	(ARM) Stream Table Entry
CD	(ARM) Context Descriptor
IDT	Interrupt Descriptor Table
ATS	(PCIe-) Address Translation Services

Appendix

Literature Taxonomy

Title	Quality Rating			
	RO	RD	FD	
0wn3d by an iPod	3	2	3	[23]
FireWire: All your Memory are belong to us	3	2	3	[25]
Hit by a Bus: Physical Access Attacks with Firewire	3	2	2	[26]
Physical security attacks on windows vista	3	3	2	[31]
Memory Forensics over the IEEE 1394 Interface	3	3	3	[29]
TRESOR Runs Encryption Securely Outside RAM	2	3	2	[33]
TRESOR-HUNT: Attacking CPU-Bound Encryption	3	3	3	[32]
I/O Attacks in Intel PC-based Architectures and Countermeasures	3	×	1	[15]
The Jedi Packet Trick takes over the Deathstar	2	3	3	[40]
Run-DMA	1	3	3	[34]
BMCLeech: Introducing Stealthy Memory Forensics to BMC	3	3	3	[36]
Direct Memory Attack the Kernel	3	3	3	[37]
Subverting Windows 7 x64 Kernel with DMA Attacks	3	3	2	[38]
Thunderclap: Axploring Vulnerabilities in Operating System IOMMU Protection via DMA from untrustworthy Peripherals	3	3	3	[39]
Taking DMA Attacks to the next Level	2	2	3	[49]
Security Analysis of hybrid Intel CPU/FPGA Platforms using IOMMUs against I/O Attacks	3	3	3	[64]

Literature Taxonomy

Title	Quality Rating			
	RO	RD	FD	
HyperLeech: Stealthy System Virtualization with minimal Target Impact through DMA-based Hypervisor Injection	3	3	3	[47]
Breaking Thunderbolt Protocol Security: Vulnerability Report	1	3	3	[48]
Inception is a physical memory manipulation and hacking tool	3	3	3	[44]
DCILeech: Leveraging Intel DCI for Memory Forensics	2	3	3	[50]
Characterizing, exploiting, and detecting DMA code injection vulnerabilities in the presence of an IOMMU	3	3	3	[70]
Thunderbolt: Exposure and Mitigation	3	×	3	[41]
Understanding DMA Malware	3	3	3	[71]
Project Maux Mk. II, I 0wn the NIC, now I want a Shell	2	3	3	[42]
Can you still trust your Network Card?	2	3	3	[43]
Kernel DMA Protection - Windows Security	3	×	2	[51]
What is Secured-Core Server for Windows Server	2	×	1	[52]
Kernel DMA Protection (Memory Access Protection) for OEMs	1	×	2	[54]
DMA protections for Mac computers - Apple Support	3	×	1	[57]
About the Security Content of MacOS Sierra 10.12.4, Security Update 2017-001 El Capitan, and Security Update 2017-001 Yosemite - Apple Support	1	×	2	[59]
An Introduction to IOMMU Infrastructure in the Linux Kernel	3	×	3	[60]
Intel® Virtualization Technology for directed I/O Architecture Specification rev 4.1	2	×	3	[61]
AMD® I/O Virtualization Technology (IOMMU) Specification rev 3.09	2	×	3	[62]

Literature Taxonomy

Title	Quality Rating			
	RO	RD	FD	
ARM® System Memory Management Unit Architecture Specification Version 3	2	×	3	[63]
Using IOMMU for DMA Protection in UEFI Firmware	3	×	3	[66]
Processor Configuration Register Definitions and Address Ranges - 1.3| 12th Generation Intel® Core™ Processor Datasheet	1	×	3	[67]
Intel® Trusted Execution Technology (Intel® TXT): Software Development Guide	1	×	2	[68]

Listing 3: Sourced Literature Material, selected and rated acc. to Methodology

List of Figures

List of Listings

List of References

[1] A. Harvey, and D. Staff, “Dma fundamentals on various pc platforms,” *Nat. Instruments, April*, vol. 8, 1991.

[2] M. Jackson, R. Budruk, J. Winkles, and D. Anderson, *PCI Express Technology 3.0*, MindShare Press, 2012.

[3] B. Dolan-Gavitt, “Forensic analysis of the windows registry in memory,” *Digit. Investigation*, vol. 5, 2008.

[4] Apple, “About the security content of macos sierra 10.12.4, security update 2017-001 el capitan, and security update 2017-001 yosemite,” 2017. [Online]. Available: https://support.apple.com/en-us/HT207615 (Accessed: 30. Jun 2023)

[5] J. Hall, A. Daken, et al., “Kernel dma protection for thunderbolt™ 3,” 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/kernel-dma-protection-for-thunderbolt (Accessed: 30. Jun 2023)

[6] F. Nocker, “Dma reloaded - building a pcileech for memory acquisition,” FH St. Pölten, 2021. (Bachelor's Thesis)

[7] A. Silberschatz, P. B. Galvin, and G. Gagne, “Operating system concepts, 10th edition,” vol. 10, p. 15, 2018.

[8] PulseSupply, “Rad data pulse supply - multi-drop polling basics,” 2007. [Online]. Available: https://web.archive.org/web/20140217174920/http://www.pulsewan.com/data101/multidrop_polling_basics.htm (Accessed: 15. Nov 2023)

[9] N. Murphy, and M. Barr, “Introduction to watchdog timers - embedded systems programming,” 2001. [Online]. Available: https://www.embedded.com/introduction-to-watchdog-timers/ (Accessed: 15. Nov 2023)

[10] J. Corbet, A. Rubini, and G. Kroah-Hartman, “Linux device drivers, third edition, chapter 10. interrupt handling,” 2005. [Online]. Available: https://static.lwn.net/images/pdf/LDD3/ch10.pdf (Accessed: 15. Nov 2023)

[11] P. Stewin, “Detecting peripheral-based attacks on the host memory,” 2015.

[12] IntelCorporation, “Thunderbolt™ 3 tech brief,” 2016. [Online]. Available: https://thunderbolttechnology.net/sites/default/files/HBD16235_Thunderbolt_TB_r05.pdf (Accessed: 15. Nov 2023)

[13] D. Abbott, “Pci bus demystified,” 2004.

[14] T. Fountain, A. McCarthy, F. Peng, and others, “Pci express: an overview of pci express, cabled pci express and pxi express,” 2005.

[15] F. L. Sang, V. Nicomette, and Y. Deswarte, “I/o attacks in intel pc-based architectures and countermeasures,” in *2011 First Syssec Workshop*, 2011, pp. 19–26.

[16] S. P. Dandamudi, *Fundamentals of Computer Organization and Design*, vol. 7, Springer, 2003.

[17] Y. Liu, D. Xu, H. Cai, and C. Yang, “System architecture design of pcie root complex based on sopc,” in *2017 36th Chin. Control Conf. (Ccc)*, 2017, pp. 5490–5495.

[18] R. Budruk, “An introduction to pci express,” 2004. [Online]. Available: http://www.mindshare.com/files/resources/MindShare_Intro_to_PCIe.pdf

[19] D. S.-C.-E. at Carleton-University, “Ppt - chapter 7 input/output powerpoint presentation,” 2007. [Online]. Available: https://www.slideserve.com/edric/chapter-7-input (Accessed: 03. Dez 2023)

[20] R. Solomon, “Pci express basics and background,” 2015. [Online]. Available: https://pcisig.com/sites/default/files/files/PCI_Express_Basics_Background.pdf (Accessed: 15. Nov 2023)

[21] M. Shona, “How to write a literature review | guide, examples, & templates,” 2023. [Online]. Available: https://www.scribbr.com/methodology/literature-review/ (Accessed: 18. Jun 2023)

[22] K. Dickersin, R. Scherer, and C. Lefebvre, “Systematic reviews: identifying relevant studies for systematic reviews,” *Bmj*, vol. 309, no. 6964, pp. 1286–1291, 1994.

[23] M. Dornseif, “0wned by an ipod,” *Presentation, Pacsec*, 2004.

[24] digitalscholar, “Zotero | your personal research assistant,” 2023. [Online]. Available: https://www.zotero.org/ (Accessed: 18. Jun 2023)

[25] M. Becher, M. Dornseif, and C. N. Klein, “Firewire: all your memory are belong to us,” *Proc. Cansecwest*, p. 67, 2005.

[26] A. Boileau, "Hit by a bus: physical access attacks with firewire," *Presentation, Ruxcon*, vol. 3, 2006.

[27] I. p1394 Working Group, and others, "Ieee standard for a high performance serial bus," *IEEE Std*, pp. 1394–1995, 1996.

[28] RiskyBusiness, "Risky business #52 – exclusive: winlockpwn code release," 2008. [Online]. Available: https://risky.biz/netcasts/risky-business/risky-business-52-exclusive-winlockpwn-code-release/ (Accessed: 20. Dez 2023)

[29] F. Witherden, "Memory forensics over the ieee 1394 interface," vol. 3, p. 2017, 2010. [Online]. Available: https://freddie.witherden.org/pages/ieee-1394-forensics/revisions/c1c615827b7647933e5a3d00668d6183.pdf

[30] OHCI, "1394 open host controller interface specification," 2000. [Online]. Available: https://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/ohci_11.pdf (Accessed: 21. Dez 2023)

[31] P. Panholzer, "Physical security attacks on windows vista," *SEC Consult Vulnerability Lab, Vienna, Tech. Rep*, 2008.

[32] E.-O. Blass, and W. Robertson, "Tresor-hunt: attacking cpu-bound encryption," 2012. [Online]. Available: https://doi.org/10.1145/2420950.2420961 (Accessed: 22. Dez 2023)

[33] T. Müller, F. C. Freiling, and A. Dewald, "Tresor runs encryption securely outside ram," in *20th USENIX Secur. Symp. (USENIX Secur. 11)*, 2011.

[34] M. Rushanan, and S. Checkoway, "Run-dma," in *9th USENIX Workshop Offensive Technologies (WOOT 15)*, 2015.

[35] U. Frisk, "Github - ufrisk/pcileech: direct memory access (dma) attack software," 2016. [Online]. Available: https://github.com/ufrisk/pcileech (Accessed: 22. Dez 2023)

[36] T. Latzo, J. Brost, and F. Freiling, "Bmcleech: introducing stealthy memory forensics to bmc," *Forensic Sci. International: Digit. Investigation*, vol. 32, p. 300919, 2020.

[37] U. Frisk, "Direct memory attack the kernel," *Proc. Defcon*, vol. 24, 2016.

[38] D. Aumaitre, and C. Devine, "Subverting windows 7 x64 kernel with dma attacks," *Hitbsecconf Amsterdam*, 2010.

[39] A. T. Markettos, C. Rothwell, et al., “Thunderclap: exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals,” 2019.

[40] A. Triulzi, “The jedi packet trick takes over the deathstar,” *Central Area Netw. Secur. (CANSEC 2010)*, 2010.

[41] S. S. John, “Thunderbolt: exposure and mitigation,” Fall, 2013.

[42] A. Triulzi, “Project maux mk. ii, i 0wn the nic, now i want a shell,” in *8th Annu. Pacsec Conf.*, 2008, p. 90.

[43] L. Duflot, Y.-A. Perez, G. Valadon, and O. Levillain, “Can you still trust your network card,” *Cansecwest/core10*, pp. 24–26, 2010.

[44] C. Maartmann-Moe, “Github - carmaa/inception: inception is a physical memory manipulation and hacking tool,” 2011. [Online]. Available: https://github.com/carmaa/inception (Accessed: 22. Dez 2023)

[45] IntelCorporation, “Intelligent platform management interface specification second generation v2.0,” 2015. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/ipmi-intelligent-platform-mgt-interface-spec-2nd-gen-v2-0-spec-update.pdf (Accessed: 27. Dez 2023)

[46] J. Sylve, “Github - 504ensicslabs/lime: lime (formerly dmd) is a loadable kernel module (lkm), which allows the acquisition of volatile memory from linux and linux-based devices [...],” 2012. [Online]. Available: https://github.com/504ensicsLabs/LiME (Accessed: 27. Dez 2023)

[47] R. Palutke, S. Ruderich, M. Wild, and F. Freiling, “Hyperleech: stealthy system virtualization with minimal target impact through dma-based hypervisor injection,” in *23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID 2020)*, 2020, pp. 165–179.

[48] B. Ruytenberg, “Breaking thunderbolt protocol security: vulnerability report,” Apr, 2020.

[49] A. Trikalinou, and D. Lake, “Taking dma attacks to the next level,” *Blackhat Usa*, pp. 22–27, 2017.

[50] T. Latzo, M. S. FAU, and F. F. FAU, “Leveraging intel dci for memory forensics,” *Forensic Sci. International: Digit. Investigation*, 2021.

[51] V. Pamnani, and P. Matarazzo, “Kernel dma protection - windows security,” 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/

security/hardware-security/kernel-dma-protection-for-thunderbolt (Accessed: 18. Nov 2023)

[52] R. Sasabe, R. Harwood, et al., "What is secured-core server for windows server," 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/security/secured-core-server (Accessed: 18. Nov 2023)

[53] K. Bridge, J. Kehr, and Others, "System power states - win32 apps," 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/power/system-power-states#working-state-s0 (Accessed: 18. Nov 2023)

[54] E. Seattle, and Others, "Kernel dma protection (memory access protection) for oems," 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-kernel-dma-protection (Accessed: 18. Nov 2023)

[55] J. Hall, D. Granito, and others, "Virtualization-based security (vbs) | microsoft learn," 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs (Accessed: 11. Dez 2023)

[56] MicrosoftSecurityTeam, "Introducing kernel data protection, a new platform security technology for preventing data corruption," 2020. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2020/07/08/introducing-kernel-data-protection-a-new-platform-security-technology-for-preventing-data-corruption/ (Accessed: 11. Dez 2023)

[57] AppleSupport, "Direct memory access protections for mac computers - apple support," 2021. [Online]. Available: https://support.apple.com/guide/security/direct-memory-access-protections-seca4960c2b5/web (Accessed: 03. Dez 2023)

[58] A. Singh, "A new screen of death for mac os x," 2010. [Online]. Available: https://web.archive.org/web/20120501191934/http://osxbook.com/book/bonus/chapter5/panic/ (Accessed: 03. Dez 2023)

[59] AppleSupport, "About the security content of macos sierra 10.12.4, security update 2017-001 el capitan, and security update 2017-001 yosemite - apple support," 2023. [Online]. Available: https://support.apple.com/en-us/103447 (Accessed: 03. Dez 2023)

[60] A. Huang, "An introduction to iommu infrastructure in the linux kernel," 2021. [Online]. Available: https://lenovopress.lenovo.com/lp1467.pdf (Accessed: 05. Dez 2023)

[61] IntelCorporation, “Intel® virtualization technology for directed i/o architecture specification rev 4.1,” 2023. [Online]. Available: https://software.intel.com/content/www/us/en/develop/download/intel-virtualization-technology-for-directed-io-architecture-specification.html (Accessed: 05. Dez 2023)

[62] AMD, “Amd® i/o virtualization technology (iommu) specification rev 3.09,” 2023. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/specifications/48882_IOMMU.pdf (Accessed: 05. Dez 2023)

[63] ArmLimited, “Arm® system memory management unit architecture specification version 3,” 2016. [Online]. Available: https://documentation-service.arm.com/static/63d7a2d5e4378a55c5e045b9 (Accessed: 05. Dez 2023)

[64] C. J. Peglow, and T. Eisenbarth, “Security analysis of hybrid intel cpu/fpga platforms using iommus against i/o attacks,” *Master's Thesis. Univ. Lübeck*, 2020.

[65] ACPI, “Acpi overview,” 2004. [Online]. Available: https://web.archive.org/web/20190525060308/http://www.acpi.info/presentations/ACPI_Overview.pdf (Accessed: 05. Dez 2023)

[66] J. Yao, V. J. Zimmer, and S. Zeng, “A tour beyond bios: using iommu for dma protection in uefi firmware,” *Intel Corporation*, vol. 1497, 2017.

[67] IntelCorporation, “Processor configuration register definitions and address ranges - 1.3 - id:767626 | 12th generation intel® core™ processor datasheet volume 2 of 2,” 2023. [Online]. Available: https://edc.intel.com/content/www/us/en/design/publications/12th-generation-core-processor-datasheet-volume-2-of-2/Chapter-2/ (Accessed: 08. Dez 2023)

[68] IntelCorporation, “Intel® trusted execution technology (intel® txt): software development guide,” 2021. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/315168/intel-trusted-execution-technology-intel-txt-software-development-guide.html (Accessed: 08. Dez 2023)

[69] ArmLimited, “Arm® system memory management unit architecture specification version 2,” 2012. [Online]. Available: https://developer.arm.com/documentation/ihi0062/dc/?lang=en (Accessed: 05. Dez 2023)

[70] M. Alex, S. Vargaftik, et al., “Characterizing, exploiting, and detecting dma code injection vulnerabilities in the presence of an iommu,” in *Proc. Sixteenth Eur. Conf. Comput. Syst.*, 2021, pp. 395–409.

[71] P. Stewin, and I. Bystrov, “Understanding dma malware,” in *Detection Intrusions Malware, Vulnerability Assessment: 9th Int. Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Sel. Papers 9*, 2013, pp. 21–41.