



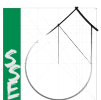
TU Clausthal



Pierre Schnarz

Security Patterns for AMP-based Embedded Systems

SSE-Dissertation 19



Software
Systems
Engineering

Institut für Informatik
Lehrstuhl von Prof. Dr. Andreas Rausch

Security Patterns for AMP-based Embedded Systems

D o c t o r a l T h e s i s
(D i s s e r t a t i o n)

to be awarded the degree of
Doktor-Ingenieur
(Dr. -Ing.)

submitted by
Pierre Schnarz
from Alzenau

approved by the Department of Informatics,
Clausthal University of Technology

2018

Dissertation Clausthal, SSE-Dissertation 19, 2018

Chairperson of the Board of Examiners
Prof. Dr. Jörg P. Müller

Chief Reviewer
Prof. Dr. Andreas Rausch

2. Reviewer
Prof. Dr. Joachim Wietzke

3. Reviewer
Prof. Dr. Jörn Eichler

Date of oral examination: December 19, 2018

Für Katrin

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or by any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Pierre Schnarz
January 2019

Acknowledgements - Danksagung

The probability that we may fail in the struggle ought not to deter us from the support of a cause we believe to be just.

Abraham Lincoln

Viele Menschen haben mich auf dem langen Weg bis zur Fertigstellung dieser Arbeit begleitet. Daher möchte ich mich hier bei all Jenen bedanken, die beigetragen haben mir dies zu ermöglichen.

Bei Prof. Dr. Joachim Wietzke bedanke ich mich für die Betreuung meiner Promotion. Gerade die Mitarbeit in seiner Forschungsgruppe und darüber hinaus hat mir die nötige Hartnäckigkeit vermittelt, welche es brauchte um dieses große Projekt zu Ende zu bringen.

Ich möchte mich außerdem bei Prof. Dr. Andreas Rausch bedanken, welcher mich in seinem Institut als Doktorand aufgenommen hat. Seine konstruktive Kritik und leitende Fragen haben mir geholfen den Inhalt dieser Arbeit zu sortieren.

Des weiteren, möchte ich Prof. Dr. Ingo Stengel und Prof. Dr. Ronald Moore danken, welche mich gerade in der Anfangsphase der Promotion begleitet haben.

Frau Prof. Dr. Elke Hergenröther und Herrn Prof. Dr. Urs Andelfinger bedanke ich mich für die Unterstützung seitens der Hochschule Darmstadt und die Chance meinen Weg über die TU Clausthal gehen zu können.

In der Zeit als wissenschaftlicher Mitarbeiter im In-Car Multimedia Labor haben mich meine Kollegen im Bereich der Technischen Informatik unterstützt. Daher möchte ich mich hier bei Bettina, Sergio, Michael, Manfred für die aufschlussreiche Zeit bedanken.

Auch meine Doktorandenkollegen in unserer Forschungsgruppe trugen einen großen Teil dazu bei, diese Forschungsarbeit zu bestreiten. Hier möchte ich Clemens Fischer danken, welcher maßgeblich an der Umsetzung der Implementierungen beigetragen hat. Tobias Holstein für konstruktive Diskussionen rund um das Thema Multi-OS. Andreas Knirsch welcher mich zur Forschungsgruppe gebracht und in all den Jahren nicht nur bei Forschungsfragen zur Seite stand. Markus Glaab mit dem ich gerade in der zweiten Phase auch gerne über die fachlichen Themen hinaus philosophiert habe.

Gerade auch meine Arbeitskollegen welche ich außerhalb der Forschungsgruppe hatte und habe, ermutigten mich bei Zweifeln und unterstützten mich bei inhaltlichen Fragen.

Über alle Inhaltlichen Aspekte hinaus, ist es mein familiäres und soziales Umfeld welches dazu beigetragen hat mich vor allem als Mensch weiterzuentwickeln.

Meiner Familie danke ich für die Stütze in allen Lebenslagen. Gerade meine Eltern Ulrike und Reiner unterstützen mich vorbehaltlos bis zur Erreichung all meiner Ziele.

Meinen Freunden danke ich für das "Da" sein, um mir einen gewissen Abstand zur Arbeit zu ermöglichen. Sabine du bist ein Freund wie ihn sich jeder wünscht.

Außerordentlich möchte ich an dieser Stelle meiner Frau Katrin danken. Mit ihrer Nähe hat sie mich und unsere kleine Familie über all die Jahre getragen und stand über den Dingen. Der Erfolg dieses Projektes geht vor allem auf sie zurück. Ihr widme ich diese Arbeit.

Abstract

The consolidation of diverse functionalities onto a single platform is an ongoing, and still emerging, trend in the development of automotive electronic control units. More and more, these software-intensive functions imply different requirements concerning their system quality. In the future, this so-called mixed-criticality systems will emerge and consolidate even more functions, towards large computational platforms. Through the advent of hardware virtualization features into automotive-grade microcontrollers, software partitioning on hardware-level has been made possible. Particularly, asynchronous multiprocessing (AMP) is suitable to host several domains "bare-metal" by utilising these hardware virtualization capabilities. The AMP paradigm aims to assign a group of hardware elements statically to a single software partition. This composition is referred to as asynchronous domain. AMP is considered to be very performance effective, while the effort of realising hypervisors is kept at a minimum. However, an important requirement of mixed-criticality systems is to provide a platform to consolidate functions with a high degree of freedom of interference, dependability and security. Particularly, the availability and integrity aspect of co-hosted functions need to be enforced. Notwithstanding the utilisation of a common hardware platform, side-effects might end up in severe vulnerabilities.

This work elaborates on security patterns considering the specific construction paradigm of AMP-based systems. The patterns include security problems and solutions describing the offensive and defensive aspects of the given context. A tailored security assessment methodology combines methods and tools to analyse, quantify and evaluate the particular artefacts. The vulnerability assessment conducted in this work revealed a surface for denial-of-service of shared last-level caches (LLC) and elevation-of-privilege and tampering threats by misusing co-processors. Accordingly, the exploitability of

these threats is demonstrated by penetration tests. The strategy to solve these issues, a reordering of the system memory map is proposed. A domain-block based mapping is shown to partition the LLC, which limits in this way the interference of adjacent domains. Furthermore, memory-map shuffling is proposed, to limit the exploitability of elevation-of-privilege threats by obfuscating the target memory structure. The findings of the security problems are transferred into rules to detect the issues in system architecture models. Furthermore, it is proposed to implement on each system layer primary and secondary security countermeasures. Particularly, systems utilizing hardware protection capabilities this leads to a extensive defence-in-depth security architecture. Therefore, the concepts contribute to the deterrence and the prevention of adverse actions to physical memory.

Table of contents

List of figures	xv
List of tables	xix
1 Introduction	1
1.1 Motivation of This Research	2
1.2 Research Goal	5
1.3 Structure	5
2 Fundamental Concepts	9
2.1 Profile of Asymmetric Multiprocessing Systems	10
2.1.1 Definitions and Terminology	10
2.1.2 ToE Description	13
2.1.3 Hardware Elements	18
2.1.4 ToE MPSoC Components	19
2.1.5 Security Problem Definition	27
2.1.6 Security Objectives	29
2.2 Security Engineering	31
2.2.1 Introduction to Security Engineering	31
2.2.2 Security Assessment Methodology	49
2.3 Research Methodology	54
2.4 Summary	59
3 Risk Assessment: Driver Information System Case Study	61
3.1 ToE- and Context Profiling	62

3.1.1	Mixed-Criticality Systems: A Case for AMP	62
3.1.2	System Decomposition	64
3.1.3	Experimental Platform	69
3.1.4	Threat Context Analysis	74
3.2	Threat and Attack Analysis	83
3.2.1	Threat Analysis	83
3.2.2	Attack Analysis	86
3.3	Risk Analysis	90
3.3.1	Impact Analysis	90
3.3.2	Attack Potential Analysis	92
3.3.3	Risk Definition	94
3.3.4	Hypothesised Attacks	94
3.4	Summary	95
4	Vulnerability Assessment	97
4.1	Hyp-Attack1: PE_i disrupts PE_j access to LLC	99
4.1.1	Vulnerability Analysis	99
4.1.2	Penetration Test: Cache Thrashing	113
4.2	Hyp-Attack2: PE 's memory base is tampered with by adjacent PE . .	120
4.2.1	Vulnerability Analysis	120
4.2.2	Penetration Test: Co-Processor Exploit	125
4.3	Generalized Vulnerability and Exploitation Pattern	131
4.3.1	Modelling Protection Architectures	132
4.3.2	Preliminaries of an AMP system	132
4.3.3	Breach Access Controls on Intermediated Level	133
4.3.4	Denial-of-Service a Shared Resource	135
4.3.5	Identification of Attack Surfaces in AMP Systems	136
4.3.6	Primary and Secondary Assets	138
4.3.7	Attack Objectives and Scenarios	139
4.4	Summary	142
5	Risk Treatment	145
5.1	Risk Treatment Strategy	146
5.1.1	Target and Residual Attack Potential	147
5.1.2	Exploitation Prevention on the Intermediate Layer	147
5.1.3	Security Solution	148
5.1.4	Primary and Secondary Countermeasures	152

5.2	Countermeasure 1: Memory Domain-Blocks	154
5.2.1	Effectiveness Requirements	154
5.2.2	Mitigation Concept Analysis	154
5.2.3	Proof-of-Concept Implementation	157
5.3	Countermeasure 2: Memory-Map Shuffling	160
5.3.1	Effectiveness Requirements	161
5.3.2	Mitigation Concept Analysis	161
5.3.3	Case Study	168
5.4	Summary	170
6	Security Evaluation	173
6.1	Domain-Block Memory Mapping	174
6.1.1	Effectiveness Assessment	174
6.1.2	Evaluation of the PoC Implementation	176
6.1.3	Residual Risk Analysis	181
6.2	Memory-Map Shuffling	183
6.2.1	Effectiveness Assessment	183
6.2.2	Residual Risk Analysis	186
6.3	Comparison to Hypervisor-based System Architectures	188
6.3.1	Attack Potential: Cache-Thrashing	189
6.3.2	Attack Potential: Tamper with Memory of Adjacent OS Guest .	191
6.4	Summary	192
7	Related Work	195
7.1	Security Requirements Engineering	196
7.2	Security Architectures of AMP-based Systems	198
7.3	Offensive Methods and Attacks	198
7.4	Exploitation Prevention	200
7.5	Summary	202
8	Conclusion	203
8.1	Contents	204
8.2	Contributions of this Research	206
8.3	Answers to the Research Questions	208
8.3.1	Research Question 1	208
8.3.2	Research Question 2	209
8.4	Limitations	209

8.5	Future Work	210
8.6	Summary	211
9	Acronyms	213
	References	221
	Appendix A Technical Specification OMAP5	237
A.1	Schematics	238
A.2	Features	239

List of figures

2.1	ToE overview.	13
2.2	Layered system hierarchy.	14
2.3	Comparison of domain separation paradigms.	18
2.4	Considered Multiprocessor System-on-Chip (MPSoC) components. . . .	20
2.5	Generalized MPSoCstructure combining diverse elements.	21
2.6	Memory architecture.	22
2.7	Communication architecture interface.	23
2.8	System memory-map concept.	26
2.9	Two-stage memory translation.	26
2.10	First and second page-table.	27
2.11	System functions with a focus on particular product qualities.	28
2.12	The fundamental dependencies of actors and artefacts in security [41]. .	32
2.13	FIRST ontology. (adopted from [49, p. 31])	37
2.14	Relationship of security metrics.	49
2.15	Risk-based Security Assessment Process.	52
2.16	Applied security evaluation process.	53
2.17	Discovering security knowledge.	54
2.18	Research Methodology.	55
3.1	Quality assignment for the Driver Information System.	64
3.2	Case Study: Driver Information System use case.	65
3.3	Technical architecture and software stack assignments.	67
3.4	Memory map overview.	68
3.5	Data flow diagram of the driver information system.	68

3.6	Exemplary E/E architecture and functional domains.	69
3.7	Texas Instruments OMAP5432 EVM [79].	70
3.8	Boot sequence of the experimental system setup.	74
3.9	Presumed path into the MC-system. ToE threats defined in 2.1.5. . . .	78
3.10	Logical Level DFD.	84
3.11	DFD of the technical platform. Dashed items are out-of-scope.	87
4.1	Considered cache thrashing scenario.	101
4.2	Cache associativity mapping. [160, p. 132]	102
4.3	Shared way-set cache principle.	104
4.4	Cache DoS attack tree.	105
4.5	Target address array allocation principle.	107
4.6	Timing diagram in flooded way-set.	108
4.7	NUMA and cache memory access latencies.	110
4.8	Sequence diagram of thrashing measurements.	117
4.9	Comparison of way-set occupancy.	118
4.10	Identity memory mapping.	119
4.11	Breach or circumvent memory protection controls in a SoC.	121
4.12	Misuse scenario utilizing a co-processor to attack a target memory area.	122
4.13	Misuse scenario utilising a GPU to attack a target memory area.	123
4.14	Implemented attack vector.	126
4.15	Memory mapping (extract) for the attack vector.	128
4.16	Exploitation firmware structure overview.	129
4.17	Exploitation firmware attack sequence.	130
4.18	System layers and entity relationships.	133
4.19	Generalized access control problem.	134
4.20	Resource sharing.	135
5.1	Primary and secondary countermeasures in relation to exploitation and asset.	148
5.2	Distributed access control.	151
5.3	Modified MPSoC architecture to prevent from direct access misuse. . .	152
5.4	Object sharing security problem model.	152
5.5	MMU utilized as means of access control.	153
5.6	Integration of a secondary countermeasure on the intermediate level. . .	153
5.7	Principle of domain blocks with exemplary mapping of main memory. .	156
5.8	Addressing of domain-blocks in main memory.	157

5.9	Principle of randomized memory assignment.	162
5.10	Integration into the boot process.	164
5.11	Coverage of the main memory shuffling.	166
5.12	Comparison of page size, shuffled memory size and the resulting entropy bits.	169
6.1	Comparison of identity and DB mapping.	178
6.2	Memory bandwidth comparison.	180
6.3	DFD of the Type-1 hypervisor system architecture.	189
A.1	TI OMAP5 IP-cores and Communication Architecture [170].	238

List of tables

2.1	Impact severity classes [67].	38
2.2	Attack Potential rating table [173].	39
2.3	Attack Potential level and likelihood [173].	40
2.4	Security risk level [67].	41
2.5	Required and Residual AP (based on [173, p. 418]).	42
2.6	Microsoft STRIDE Threats. [154]	42
2.7	Data flow diagram	43
2.8	Threats per Data Flow Diagram (DFD) Element.	43
2.9	CVSS Base Score [47].	46
2.10	CVSS Base Score continued [47].	47
3.1	Listing of logical level threats, including a description of the violated security goals.	85
3.2	Technical DFD: STRIDE-per-element. Check marks in brackets state out-of-scope threats.	87
3.3	STRIDE-per-element analysis.	89
3.4	Impact analysis (S afety, F inancial, O perational, P rivacy).	91
3.5	Attack: PE s memory base is tampered with by adjacent PE	92
3.6	Attack: PE_i disrupts PE_j access to LLC	93
3.7	Risk analysis results.	94
4.1	Caching terminology and parameters.	103
4.2	Symbols for Cache Scheduling.	108
4.3	Symbols of cache access latency metrics.	110

4.4	CVSS base score of the cache-thrashing exploit concept.	111
4.5	Measurement symbols.	114
4.6	Overview of the measurement results.	118
4.7	Common Vulnerability Scoring System (CVSS) temporal score of the Last Level Cache (LLC) disruption pentest.	119
4.8	CVSS base score of the access control breach.	124
4.9	CVSS temporal score of the co-processor misuse pentest.	130
5.1	Tactics for threat mitigation (summary of [154, p. 145ff]).	150
5.2	Domain-block specific parameters. In addition to Table 4.1	157
5.3	Platform cache parameters.	159
5.4	Entropy and probability symbols.	166
6.1	CVSS base score of the domain-block concept.	175
6.2	Effectiveness Domain Block concept.	176
6.3	Measurement symbols.	177
6.4	Overview of the measurement results.	178
6.5	CVSS Environmental score of the Domain-Block mapping.	181
6.6	Attack: PE_i disrupts PE_j access to LLC with applied Domain-Block mapping.	182
6.7	Domain-Block Mapping: residual risk analysis results.	183
6.8	CVSS base score of the memory-map shuffling concept.	184
6.9	Effectiveness memory-map shuffling countermeasure.	185
6.10	CVSS Environmental score of the Memory-Map Shuffling.	186
6.11	Modified Attack Potential: PE_i is tampered with by adjacent PE with applied Memory-Map Shuffling.	186
6.12	Residual risk analysis results.	188
6.13	Hypothesised attack: OS_{guest} disrupts adjacent $OS_{guest}'s$ access to LLC	190
6.14	Hypothesised attack: OS_{guest} is tampered with by adjacent OS_{guest}	191

Contents

1.1	Motivation of This Research	2
1.2	Research Goal	5
1.3	Structure	5

1.1 Motivation of This Research

Complex systems such as automobiles consist of a plethora of Electronic Control Unit (ECU)s. The number of such embedded devices ranges from 60 over to 100, depending on the particular model [117]. Traditionally, each of those ECUs were dedicated to a specific purpose, meaning roughly to be categorized into sensors, actuators and processing units. For example, the brake controller actuates on the de-acceleration of the vehicle, while the cluster instrument processes and provides a visualization of the driving information to the driver, as well as, the so-called head unit that processes the infotainment features and provides it to the user. However, the consolidation of functions into a single system platform (such as an ECU) is an emerging trend in automotive system engineering [90]. The vehicle Electric/Electronic (E/E) architecture has changed from many-single purpose ECU networks to a few so-called centralized Domain Control Unit (DCU)s [76, 162]. This might be motivated by the need to efficiently implement driver assistance features such as the Electronic Stabilization Program (ESP) at a centralized point within the vehicle. Furthermore, Advanced Driver Assistant Systems (ADAS) features aiming at the automotive megatrend, automated driving, imply much more fusion and processing of data [34].

The challenge induced by this consolidation is to consider the diverse quality goals of the different functions and features. To refer back to the previous example, there are functions obviously dealing with safety of the driver, such as the brake controller. Accordingly, the reliability of that function is crucial not only to the driver itself but also for many stakeholders. In contrast to that, infotainment features demand processing power in order to embellish the user experience. Consequently, the supply chain for these domain controllers is differently facilitated. In the old-fashioned landscape, each supplier delivered a system consisting of the hardware and software to the customer (Original Equipment Manufacturer (OEM)) who is integrating it into the vehicular network. Considering domain controllers, the hardware is supplied by one party and the software that implements the functions is delivered by third parties. This challenges the trustworthiness of each delivery [19]. As a result, domain controllers are multi-tenant Mixed Criticality Systems (MC-systems). They aim for function consolidation while keeping them separated as they were before in dedicated ECUs.

By introducing the connected-car, automobiles have become the next *thing* in the Internet-of-Things (IoT). An increasing connectivity of the vehicle will transfer automobiles from smart cars to self-driving (autonomous) cars [125]. The possibilities of communication to either, other vehicles (Vehicle-to-Vehicle (V2V)), to the infrastructure

(Vehicle-to-Infrastructure (V2I)) or to online (cloud) services are tremendous. Many value-adding online services or a virtual 360-degree view will be made possible. However, along with the increasing connectivity, a number of security incidents are observable. This is particularly caused by the increased attack surface due to the new, remotely accessible interfaces. Researchers began in 2011 to conduct surveys to turn out the attack surfaces of modern automobiles [25, 121]. The wireless entry vectors for adversaries are broad, ranging from near field communication technologies such as Bluetooth over to Long Term Evolution (LTE), wide range, technologies. Recently, it has been demonstrated that those services are exploitable in order to introduce a safety-relevant incident [59, 122, 158]. These grounds finally gave a rise to the need of sophisticated security engineering within the automotive development [137]. Meaning, former assumptions are made invalid. Attack surfaces go extensively far beyond direct physical access to the vehicle [25]. The benefit of bringing rigidly implemented security engineering into the development cycle will help to increase the security of products [176].

Bringing the previous arguments together, there are efforts to consolidate functions into powerful domain controllers inside the vehicle. In parallel, features of the vehicles have lead to new mega trends in mobility. This is not only grounded in connectivity, but to a significant extent driven by it. On the other hand, this also brings stakeholders to the field who wish to misuse those developments for their own purposes. This is why making considerations to security in those areas is necessity.

So, it is reasonable to assume that adversaries have found their way into a single system, such as an ECU in a vehicle. It is then worthwhile to raise the question: how does a compromisation from the outside propagate through the entered system? What steps are necessary for the attacker in order to reach his goal? The goal, in this case, is something they want to gain benefit from. From the protection perspective, the aim is to make it *hard* or *harder* to take a further step into the system. Furthermore, the number of steps to take shall be as high as possible, which is the core idea of defence-in-depth [7]. That means hurdles will be integrated that facilitate such previously mentioned steps. Concepts, such as isolation or separation, address this aim. Those concepts aim for the effort to prevent one entity from interfering with another. As an example, particularly in automotive systems, a function's security properties such as the integrity and availability will need to avoid external adversarial actions, in order to reduce the risk of safety impacts. A practical example of isolation mechanisms are network firewalls that separate network segments from each other.

How those isolation mechanisms are facilitated depends on the system and its technical architecture. From this technical viewpoint, isolation of functions can be achieved on several levels of the system architecture. These levels apply to both the software-stack and the hardware. For example, software levels are often differentiated by privilege levels. Here, processes which represent the actual functions, operating systems which manage the resources for the processes and the hardware, are worth mentioning. If multiple Operating Systems (OSs) are applied, then virtualization comes into play. Meaning, a further layer is added to the system levels. Particularly, virtualization is not new to the automotive embedded systems domain [126]. However, the specific implementation experienced new opportunities due to the integration of virtualization capabilities into the embedded micro-controllers.

Asymmetric Multiprocessing (AMP) is a way to approach isolation by utilizing dedicated hardware capabilities. This enables the separation of several software-stacks, such as OS, on top of it. In contrast to traditional virtualization schemes, whose involve a virtual machine monitor or a hypervisor, the AMP approach lacks this kind of software instance between the multiple OS and hardware [30]. The means of indirection, which is enforcing the isolation, are distributed into the hardware elements. AMP aims at separating resources on the intermediate level between the software-stacks and the hardware elements. As a result, the levels above have an asymmetric access to the hardware resources [108, 109]. By applying the concept of AMP in embedded systems, it experiences a revitalization from the earlier years of large-scale main-frame systems when the OS capabilities of handling several processors on a single hardware were limited. In the field of software engineering, particularly in the automotive area, there are advantages of applying this concept. For example, OS running on top of the AMP system does not need to be modified to run on a hypervisor. Accordingly, the development cost is reduced and the maintenance effort is limited. A further, very substantial aspect in embedded systems is the performance, in this case, more precisely, the negative performance impact by introducing a further level of indirection. AMP systems run the software-stacks directly on the hardware, rather than implement the indirection in software that has, in turn, to be handled by the processor instead of doing other computations [63, 82, 131].

Embedded system hardware is often facilitated by the concept of MPSoCs. This is particularly true for automotive ECUs. MPSoCs combine processors, peripherals and memory through a particular infrastructure to a heterogeneously integrated circuit [75]. As it was mentioned previously, the isolation of software-stacks in AMP systems is moved to the hardware. Since MPSoCs can be seen as a network of hardware elements,

the challenge in isolation is to handle this complexity and apply the protection to these elements.

Isolation instantiates aspects that contribute to the security of systems, in this case, of AMP systems. To come up with means to implement protection for isolation is subject to structured security-enabled development processes. However, the continuous character of security has to be taken into account. Assuming there is any inconsistency or vulnerability in the design of isolation, the question arises as to how to handle the immutability of the hardware protection architecture. Despite that the hardware manufacturer might come up with new architecture to fix the weakness, if the system is already deployed, changes to the hardware are impossible or at least very cost intensive. As a result, if the protection mechanisms are moved to the hardware, there is no further line of defence in the case of compromise. The consequences could be cumbersome, depending on the particular use-case of the system. In order to solve this issue, several strategies are possible. One option is to put an effort into hardening another hurdle at a higher level in order to increase its effectiveness and keep the adversary out of the system. This means the problem is solved at a different place in the environment. If the development is still in progress, one might put an effort into hardening the AMP isolation as it is. For example, this could be done through rigid and formalized development, meaning, it is tried harder to avoid any vulnerabilities.

Nonetheless, the specific issues of AMP system security could also be investigated. This is then applicable to countermeasures that use the AMP specific construction paradigm again to limit the potential of the system of being attacked.

1.2 Research Goal

New paradigms of building systems might require rethinking assumptions made from traditional paradigms. This work seeks to build security patterns for AMP system paradigms. The security pattern shall describe¹ recurring security problems in a particular context and conclude with a security solution. The given concepts shall suitably comply with constraints given by the automotive context.

1.3 Structure

The structure of this thesis is outlined as follows.

¹according to Schumacher et al. [149]

Fundamental Concepts: In Chapter 2, the foundations and state-of-the-art information according to the research area is given. It profiles the Target of Evaluation (ToE) and differentiates other, recently applied paradigms. Furthermore, the means of security engineering are elaborated. This concludes with a description of the security process employed in this work. On this basis, the research methodology is described. This includes the research questions, approaches, methods and artefacts conducted throughout this work.

Risk Assessment: Driver Information System Case Study: In Chapter 3, the foundations are turned into a case-study showing the application of the previously introduced concepts. Here, a driver information system instantiates an MC-system based on the AMP paradigm. The case-study is intended to provide a context and running example in the thesis. Furthermore, the experimental platform that has been used for several activities throughout this work is explained. The security assessment derives two hypothesised attacks that are investigated further in this document.

Vulnerability Assessment: Chapter 4 conducts the vulnerability assessment. Here, the specific security aspects of the hypothesised attacks from the case-study are assessed. Furthermore, the findings are abstracted to a general attack model for the given ToE class.

Risk Treatment: Measures against the identified findings from the previous chapter are described in Chapter 5. Two countermeasures that address the security issues from the vulnerability assessment are examined. The findings are derived from a general protection model that categorizes the dependencies between the attacks and countermeasures.

Security Evaluation: The security evaluation is conducted in Chapter 6. The attacks and countermeasures have been scored against a common scheme. In this chapter, the effectiveness based on the common scoring scheme is elaborated upon. Furthermore, a brief comparison to a hypervisor based system is given. Here, the findings also are discussed by evaluating the solutions in the context of the case-study.

Related Work: In Chapter 7, the findings are aligned with related work in the particular research fields this work deals with.

Conclusion: This document concludes with Chapter 8, where a summary of the key contributions and findings of this research is presented. Furthermore, an outline of future work is given.

Fundamental Concepts

Target of Evaluation, Security Process and Research Methodology

Contents

2.1	Profile of Asymmetric Multiprocessing Systems	10
2.1.1	Definitions and Terminology	10
2.1.2	ToE Description	13
2.1.3	Hardware Elements	18
2.1.4	ToE MPSoC Components	19
2.1.5	Security Problem Definition	27
2.1.6	Security Objectives	29
2.2	Security Engineering	31
2.2.1	Introduction to Security Engineering	31
2.2.2	Security Assessment Methodology	49
2.3	Research Methodology	54
2.4	Summary	59

Out of the crooked timber of humanity, no straight thing was
ever made

Immanuel Kant

Discussing the security of a system requires a comprehensive definition of the ToE. Particularly, AMP systems rely on the hardware architecture and composition of hardware elements. This diversity needs to be abstracted in order to handle the complexity on the one hand and make results transferable on the other. In addition, the term *security* is overused in many contexts. It is mandatory to discuss what it means for the intended purpose of this work.

Accordingly, this chapter addresses these two significant issues. It first profiles AMP systems, which facilitates the ToE considered by this work. This definition of how the AMP paradigm is treated is supported by the reflection of the state-of-the-art and differentiation to other system paradigms. The second part provides an overview of how to address security in systems engineering, and concludes with the security assessment methodology applied in this work. Given that foundation, the chapter ends with the description of the research methodology including the research questions, methods and artefacts.

2.1 Profile of Asymmetric Multiprocessing Systems

2.1.1 Definitions and Terminology

Asset: Anything that has value to an owner.

Threat: Potential cause of an unwanted incident, which may result in harm to a system or organization [86].

Threat Agent / Attacker: Individual who acts adversely on an asset.

Attack: Attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized [86]. use of an asset.

Risk: Effect of uncertainty on objectives [86]. Quantitative estimation of an event (attack), with respect to its impact and likelihood to occur.

Impact: Negative effect which a successful attack causes.

Attack Potential: The Attack Potential (AP) quantifies the "*effort required to create the attack*"[173, p. 420].

Risk Treatment: Process to modify risk [86].

Risk Mitigation: A mean which reduces the attack potential and, therefore, the risk of an asset.

Countermeasure: The implementation of a risk mitigation.

Security function: Synonym to countermeasure. Implements measure against adversarial actions.

Vulnerability: Weakness of an asset or control that can be exploited by one or more threats [86].

High Level Security Goals: Confidentiality Integrity Availability (CIA) are the most common and fundamental security goals. Additionally, authenticity and authority are properties a common embedded system setup has to deal with.

The action of acting adversely on a target is referred to as attacking or compromising. The term attack defines a series of actions that achieve a particular objective. The victim of an attacker is referred to as the attack target. In a multi-level system architecture, this includes processes, threads, operating systems, Asynchronous Domains (ADs), etc. For example, an attacker may act adversely on a target AD or a target process, respectively. Reaching the phase of a successful attack, the target's state is considered as being compromised, illicit or malformed. Accordingly, the target's state is considered as being untrusted which is in contrast to the trusted state from before. An attack is usually mounted from a malformed origin, a compromised source or an illicit entity.

Definition and Operational Usage

The consideration of this work aims for the particular instantiation of separation in an AMP system. AMP describes the paradigm to enforce a spacial distribution of software functions on a common computing platform. Within this work, AMP is referred to as¹:

Definition 2.1 *AMP is a system utilization paradigm aiming for a spacial separation of hardware elements in order to establish ADs on the software layer.*

As a result, an AMP system consists of two or more ADs. To address the logical composition of several hardware elements and software components, the term AD is defined.

¹System separation paradigms are elaborated in detail in Section 2.1.2

Definition 2.2 *An AD is a logical composition of software components ran by the same processing element (Central Processing Unit (CPU) core). AD are decoupled from adjacent domains by definition.*

Typical uses of an AMP-system are highly integrated systems that aim to combine diverse functions in heterogeneous software-stacks. Accordingly, AMP is a paradigm to organise computer systems such as embedded, desktop and server systems. The intention of facilitating an AMP system has changed over the last decades of computer history. In the early days of mainframes, AMP was used as a stopgap, due to the inability to manage multiple processors within a single operating system. Today's definitions of AMP systems comply to the heterogeneous nature of MPSoCs. Colin [30] defines AMP as a system consisting of multiple processors, each of which can apply a different architecture. Each processor operates on its memory space, while an operating system is not mandatory. Furthermore, communication facilities between the processors are applied. A more technical view on the AMP-paradigm is given by ARM in the technical documentation:

"An Asymmetric Multi-processing (AMP) system enables you to statically assign individual roles to a core within a cluster so that, in effect, you have separate cores, each performing separate jobs within each cluster. This is known as a function-distribution software architecture and typically means that you have a separate OS running on the individual cores. The system can appear to you as a single-core system with dedicated accelerators for certain critical system services." [109, p. 14-7]

In Chapter 3, the concept is applied to an exemplary automotive embedded system.

ToE Type

The ToE facilitates the software configuration of a hardware platform. Throughout this work, this facilitation is referred to as the *intermediate layer* between the physical hardware and the logical software-stack(s). As a result, it is considered how the hardware is to be interfaced. There is a direct dependency on the composition of logical software-stacks (ADs) and the composition of hardware elements. In the following sections, the building blocks that are considered in this work are elaborated upon. The hardware platform consists of integrated building blocks. In this work, MPSoCs for embedded environments are considered. Generally, these components include processors and peripherals, for example. The MPSoC elements are considered to be shared by

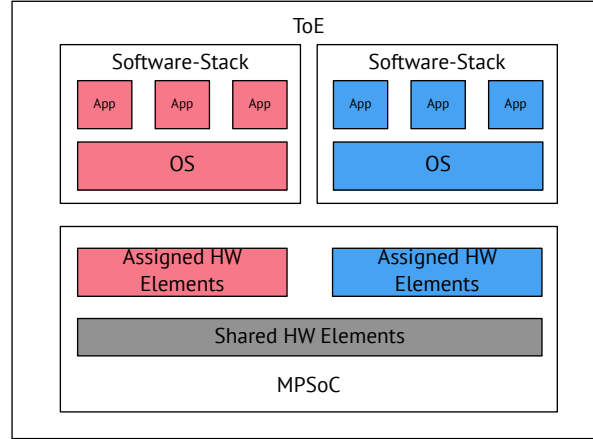


Fig. 2.1 ToE overview.

all ADs (referred to as *Shared HW Elements*) or they are assigned to a specific AD (referred to as *Assigned HW Elements*). Figure 2.1 shows a brief overview of the ToE architecture. The figure shows two ADs coring the assigned software-stacks and assigned HW elements in *red* or *blue*, respectively.

Non-TOE Software Applications and Functionality

In this work, particular features and functionalities that are run by the described ToE are not considered. This enables a context-free conduction of results. Furthermore, as it is mentioned in the previous section, only the interface between the hardware and the software is concerned. Particular OSES or applications that are run by such an AMP-based ToE are out-of-scope. For example, it will not matter if the OS implements a microkernel or monolithic-kernel architecture.

2.1.2 ToE Description

Architecture

System Privilege Hierarchy The ToE consists of several logical ADs that are a collection of software-stacks and physical hardware elements. In general, the considered ToE architecture is comprised of two main layers: the *software-stack* and the *SoC Layer*. It is considered that the hardware allows to run software in the software-stack in more than one privilege levels. This is needed to allow, privileged software to separate less privileged software components from each other. In order to implement an AMP system, MPSoC elements need to be assigned to a particular AD. Accordingly, the

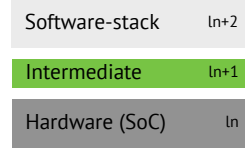


Fig. 2.2 Layered system hierarchy.

layer that separates the several MPSoC elements and assigns them to software-stacks must run on a higher privilege level than the software-stack. As a result, this is referred to be run on the *intermediate layer*. A visualization of the layers is depicted in Figure 2.2.

Layering in general aims at maintaining the divide-and-conquer principle to handle complex systems. Therefore, in the following, the platform layer structure of a system platform is given. Architectural layers aiming at dividing complex systems horizontally. This makes it possible to distinguish specific aspects within several layers from each other. Particularly, in security assurance methodologies a layered consideration is recommended. A prominent example is the Common Criteria (CC) security evaluation framework [172]. To conclude, layering will provide the foundation for the security consideration within this work. In the following, the particular layers are introduced.

SoC layer: The System-on-Chip (SoC) layer is considered to instantiate hardware layer. Typical entities are the distinct building blocks of that a SoC consists of. These building blocks are commonly referred to as Intellectual Property (IP) blocks, or cores. IP blocks are, by their nature, pre-qualified and integrated into the SoC layout. A minimal set of IP blocks includes a Processing Unit (PU), Read Only Memory (ROM), and Random Access Memory (RAM). MPSoCs integrate more than one PU. The communication is facilitated by simple protocols such as Serial Peripheral Interface (SPI) [106] or I^2C [106, 151] or complex communication infrastructures such as the ARM Advanced Microcontroller Bus Architecture (AMBA) and Advanced eXtensible Interface (AXI) bus [11]. The IP blocks are considered to be configurable via a management interface. Technically, this is represented by memory registers mapped to the system memory address space. This is elaborated upon in the following sections on the practical implications of AMP systems. Furthermore, a security-relevant configuration can be applied, such as access control to the IP blocks.

Intermediate layer: The intermediate layer represents the link between the hardware layer and the software layer. Entities on this layer are distinct software

stacks. For example, these include operating systems (OS) of any kind or more straightforward constructs such as firmware. To create these entities, the configuration interface of the SoC layer has to be utilised. Typical configuration interfaces include: translation tables for the Memory Management Unit (MMU)s to apply access control and proper function of the memory accesses, interrupt tables to facilitate signalling between the PU and power control for energy and frequencies to configure the appropriate clocks for the hardware elements. The intermediate layer is sometimes referred to as hypervisor layer or the layer at which the Virtual Machine Monitor (VMM) is implemented.

Software-Stack layer: On the software-stack layer, everything is included that runs on top of the hardware elements. Typically, this includes an OS representing the lowest logical layer of the software stack. Entities on this layer are typically the infrastructure to manage the operation of applications. Usually, this infrastructure is referred to as the OS kernel. In general, two types of kernels are differentiated: micro-kernels where the infrastructure runs as much as possible in small and self-contained components and monolithic kernels that integrate the entire functionality into a common, so-called, kernel-space. To fulfil extensive requirements of safety-relevant or multimedia applications, different types of operating systems are used. These include, Real-time Operating System (RTOS), General Purpose Operating System (GPOS) or Mobile Operating System (mobileOS). On top of the OS, the functional logic of the system is encompassed. Features and applications of the system are implemented at this logical application layer. Technically, the layer entities are represented as processes or threads that are maintained by the OS. For example, Linux OS consists of the OS layer and the application layer. The latter is called user-space and the former kernel-space. Despite the separation of user-space and kernel-space privilege granularity, processes can be assigned to access control compartments with fine-grained privileges.

Differentiation of Separation Paradigms This section seeks to differentiate the AMP-idea from other paradigms to partition software-intensive systems. Partitioning, or separation, is achievable on all layers of a system architecture. An important aspect is the utilization and management of the hardware. This influences at which layer the special separation must be implemented.

Symmetric Multi Processing. Symmetric Multi Processing (SMP) systems are the pendant to AMP systems. Here, only one software-stack instance is controlling the hardware, symmetrically. SMP systems apply multiple processors as well, but they all operate in the same memory space. Typically, a single instance OS is used to run the sharing all processors [30]. The vast majority of computer systems utilize the SMP paradigm. The infrastructure of a single OS is managing the underlying resources and makes them available for applications to the upper system levels.

"[...] SMP is a software architecture that dynamically determines the roles of individual cores. Each core in the cluster has the same view of memory and shared hardware. Any application, process, or task can run on any core, and the operating system scheduler can dynamically migrate tasks between cores to achieve optimal system load. A multi-threaded application can run on several cores at once." [109, p. 14-3]

Virtualization Virtualization, in general, is a term vastly used in the IT and desktop computing area. Silberschatz et al. defines virtualized systems as follows.

"Generally, with a virtual machine, guest operating systems and applications run in an environment that appears to them to be native hardware and that behaves toward them as native hardware would but that also protects, manages, and limits them." [155, p. 711]

The main purpose is to run multiple operating systems on a single system. In literature, a virtual machine monitor (VMM) or hypervisor facilitate the infrastructure to create virtual machines.

"A relatively small control program called Virtual Machine Monitor (VMM) or Hypervisor is placed between the OS and the hardware. Typically the VMM executes in privileged mode and can host one or more operating systems – GuestOSs – in a sandbox called Virtual Machine: a controlled construct of the underlying hardware." [120, p. 2]

As a result, virtualization endeavours to put a privileged instance underneath guest OSs in order to manage their access to the underlying hardware. These instances are commonly referred to as hypervisor or VMM. The community differentiates three types of facilitating virtualization [155].

Type-0: The virtualization capabilities are incorporated into the hardware. The virtualization infrastructure is managed by a configuration interface.

Type-1: A *type-1* virtualization runs the hypervisor bare-metal on hardware.

Type-2: A *type-2* solution relies on a so-called host OS that manages the system abstraction. The host OS provides the infrastructure to virtualize the guest OS.

Furthermore, it is differentiated between para-virtualized systems and full-virtualized systems. The former considers adaptations of the guest OSs to the underlying virtualization solution in order to make it functional or to improve performance. The latter aims to fully abstract the hardware by the hypervisor.

Type-0 Virtualization and AMP In this work, Type-0 virtualization capabilities of the hardware are treated as a precondition in order to implement an AMP system. AMP describes the aim of dividing hardware elements into ADs. Therefore, the hardware, or the particular elements of the hardware, must support assignments to logical compositions. The practical implications and the interface of recent hardware architectures are elaborated in Section 2.1.3.

"Implementing efficient virtualized systems cost effectively requires hardware support. In particular memory management can provide great challenges and have severe repercussions on system reliability and performance. To address this ARM is introducing the Virtualization Extensions to its ARM v7 architecture and the System Memory Management Unit (SMMU) Architecture." [120, p. 1]

Comparison of Domain Separation Paradigms Vertical separation software-stacks is a well-established technique to organise software entities in a software-stack. In the following, it is briefly shown how AMP relates to other partitioning schemes. In general, three schemes are differentiated. Figure 2.3 depicts three paradigms. The architectural system layers are visualized according to the multilayer scheme shown in Figure 2.2. The separated logical domains are coloured in *blue* or *red*, respectively. On the left, the SMP concept is depicted. It shows one *domain* that spans over the entire *software-stack* as well as the entire *hardware*. Accordingly, the software-stack controls the entire hardware symmetrically by privileged software that is included into the software-stack. In contrast to that, in the right example, AMP is depicted. Here, two domains span over dedicated *software-stacks* and assigned hardware *elements* of the MPSoC. These elements are considered as residing on the intermediate layer. In the centre, domain separation using a hypervisor on the intermediate layer is shown.

Here, the *hypervisor* abstracts the underlying hardware and provides a context for the software-stacks on top of it. In this way, two separated domains are created.

In Figure 2.3, the three approaches to create software partitions are shown.

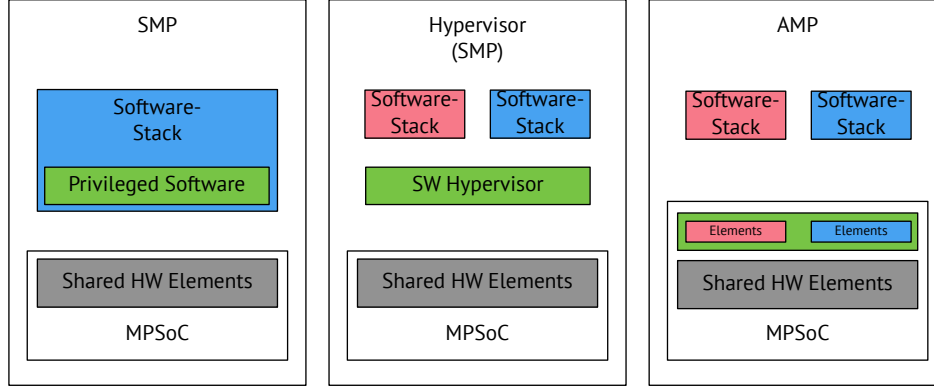


Fig. 2.3 Comparison of domain separation paradigms.

2.1.3 Hardware Elements

MPSoC fit well into the heterogeneous environment such as the automotive. Accordingly, Wolf et al. defines MPSoCs as follows:

"Multiprocessor systems-on-chips have emerged in the past decade as an important class of very large scale integration (VLSI) systems. An MPSoC is a system-on-chip—a VLSI system that incorporates most or all the components necessary for an application—that uses multiple programmable processors as system components. ... They are not simply traditional multiprocessors shrunk to a single chip but have been designed to fulfill the unique requirements of embedded applications." [186, p. 1701]

Basically, two general types of MPSoCs are differentiated in literature: heterogeneous and homogeneous architectures [75]. The former is composed of diverse system components such as processors, memories, accelerators and peripherals. The latter refers to the multiple instantiation of the identical processor system component [179]. In many cases, the heterogeneous approach fits well with automotive needs. In this case, heterogeneous means that a combination of different processor architectures can be possible. Accordingly, plenty degrees of freedom regarding the composition of functions are possible. For example, a low energy, real-time capable architecture can be combined with a high-performance general-purpose processor. Furthermore, the integration of

pre-qualified software can easily be migrated to the platform by adding the suitable processor architecture to the MPSoC. To this end, the high-level integration demands are supported by the hardware platform as well.

Wolf et al. analysed in [186] the technological history of MPSoC and states that the most MPSoC architectures follow a heterogeneous structural taxonomy. The authors argue that the architectural design is mostly influenced by the application of the hardware. Nevertheless, standards can influence the MPSoC structure as well as the centralization of a particular system component. In such cases, often a communication architecture centralised approach is realised [186] .

Many SoC vendors establish platform-based approaches to foster their product-lines. Usually, the platforms are tailored to a specific application area, such as in-vehicle infotainment systems, including a certain amount of variability to fit the product to the customer's needs. The variations reach from adding more processor cores, more memory or different accelerators such as Digital Stream Processor (DSP) or Graphical Processing Unit (GPU)s. As a result, the SoCs emerge from very specialized to commodity platforms based on reusable components [22].

Practical, state-of-the-art and automotive relevant examples include the MPSoC platforms such as Texas Instruments *OMAP*, Freescale *iMX.X*, Xilinx *Zync*² and Renesas *RCar*³.

2.1.4 ToE MPSoC Components

The ToE consists of a particular set of typical MPSoC components that are presented in this section. In general, this work considers MPSoCs as follows:

"They combine several embedded processors, memories and specialised circuitry (accelerators, I/Os) interconnected through a dedicated infrastructure to provide a complete integrated system. Contrary to SoC s, MPSoCs include two or more master processors managing the application process, achieving higher performances." [75, p. 1]

Accordingly, MPSoCs facilitated by the ToE integrate diverse system elements such as Processing Element (PE), Memory Architecture (MA), Communication Architecture (CA) and further peripherals. Particularly, heterogeneous MPSoCs integrate diverse PEs each of which is intended for different purposes. Sometimes these heterogeneous processor integrations are also referred to as multi-core or many-core SoC. Viewing

²<http://www.xilinx.com/products/silicon-devices/soc.html>

³http://am.renesas.com/applications/automotive/cis/cis_highend/index.jsp

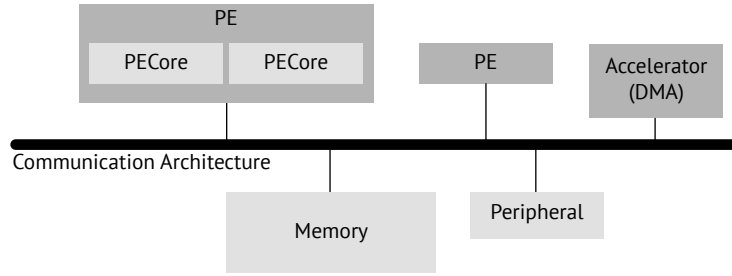


Fig. 2.4 Considered MPSoC components.

from the perspective of *Flynn's Taxonomy*, the general architecture is based on the Multiple Instructions Multiple Data (MIMD) architecture [167]. Multiple processors work independently and asynchronously on different data sets while sharing a common hardware platform. In Figure 2.4 an overview over the considered SoC elements is depicted. These elements are described in detail in the following sections.

Processing Elements

The term PE will be used as a general-purpose abstraction for active system elements or IP-cores. Active means, in this context, that the PEs have their independent and asynchronous control flow. Usually, this control flow is defined by machine code that the processor is operating on. CPU is a typical example of a PE. Nevertheless, the PEs does not necessarily need to impersonate a processor in a general meaning. Co-processors, accelerators such as GPUs or DSPs which usually support the PE by special purpose computing results. The heterogeneous nature of MPSoCs allows for the combination of different processor architectures. Those include Reduced Instruction Set Computer (RISC) architectures, stream processors (such as GPU), DSP and Field Programmable Gate Array (FPGA). A widely used representative is the ARM architecture. Nevertheless, real-time affine architectures such as the *SH-4* or PowerPC can be included as well. As an example, the Renesas *R-Car H3* platform provides a *SH-4* system component within their automotive infotainment portfolio.

The internal components of the PE depend on their specific architecture. Most commonly, they integrate a processor core and infrastructure to handle data/instructions and addresses. Particularly, the MMU plays an important role in redirecting address requests from PUs⁴. In Figure 2.5, a two-stage address translation is depicted. Furthermore, the processor caches represent a key element of a PE. Particularly from the performance perspective, characteristics such as size, organisation and relationship

⁴The MMU will be elaborated in more detail in Section 2.1.4

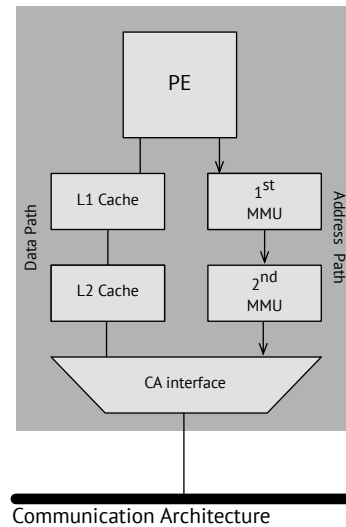


Fig. 2.5 Generalized MPSoC structure combining diverse elements.

to the cached memory are key factors. Modern PEs operate on a two-levelled cache subsystem. Level one is typically small in size and usually is split into an instruction cache and data cache (I-cache and D-cache), while for the second level, the instructions and data are mixed and the cache size is significantly larger. At the end of the line, the PE is connected by an interfacing element which enables the access to the communication architecture. This concept is depicted in Figure 2.5.

Also, the PE might consist of multiple processing cores sharing a certain part of the caching and addressing the infrastructure of the hardware element. Typically, the level one cache is private to each processing core whereas the second level is shared by all cores. In theory, the number of processing cores is not limited. As of today, the typical number of cores of a multi-core PE ranges from two (dual-core) to eight (octa-core) cores.

Memory Architecture

The memory hardware element is a crucial part of the MPSoC architecture. In general, the memory is considered to implement the RAM, which stores the instructions and data to be operated by the PEs. Upon request, the memory subsystem provides read or write operations to access the data in its physical storage over an interface. Physically, the memory is considered a volatile memory. The concept of integrating a shared memory element relates to the concept of Unified Memory Architecture (UMA). However, this would then lead to a SMP system. In order to achieve an AMP system,

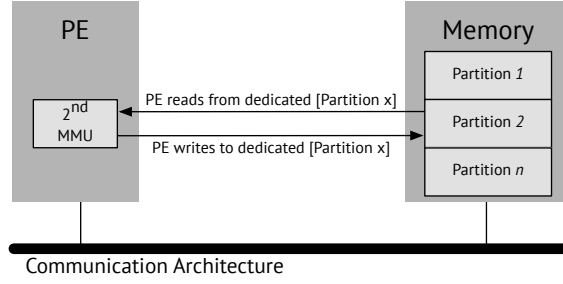


Fig. 2.6 Memory architecture.

hardware capabilities need to be utilized to dedicate memory partitions to particular ADs.

Recent MPSoC implementations integrate Double Data Rate (DDR)-Synchronous Dynamic Random Access Memory (SDRAM) for example in SoCs such as shown in [169]⁵. However, the particular physical implementation of the memory storage is out of the scope of this work. Therefore, it is considered as being abstracted by the CA interface of the memory element. Furthermore, the memory is considered to be passive on memory requests. By providing a memory address, the interface will retrieve or store data to or from the physical storage. In Figure 2.6, the model is depicted.

Communication Architecture

The communication network connects all hardware elements (IP-cores) of the SoC. Therefore, it plays a central role in the communication architecture of the system. Technically, there are numerous topologies, protocols and utilisation strategies to implement communication architectures for SoCs [123]. The topologies reach from communication bus approaches up to complex Network-on-Chip (NoC) architectures. The selection of a certain topology depends either on bandwidth requirements, system complexity or scalability [65]. NoC approaches are packet-based comparable to computer networks. Through their switching and routing facilities, NoCs are said to be scalable, while bus-based communication architectures are cost-effective yet limited in scalability.

Representative examples of CAs applying a bus topology have been conducted by Mitic et al. [123]. For Example, the authors introduce bus systems such as the ARM AMBA [11, 159] and the IBM CoreLink [78]. NoC CAs are investigated by Bjerregaard et al. [15]. Among the capability to connect IP-cores, the CA can be used to synchronise data retained in the cache systems of PEs. The ARM AMBA

⁵The experimental platform used in this work bases on the mentioned memory architecture

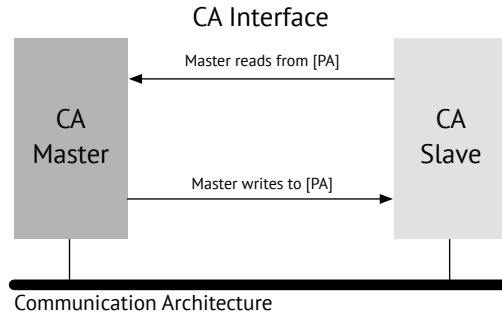


Fig. 2.7 Communication architecture interface.

AXI Communication Network incorporates such a last level cache coherency protocol [161]. Sharing data across hardware element borders can now be handled without the involvement of a dedicated memory system component. This technique enables low latency and high bandwidth data transfers.

For this work, an interface is assumed which abstracts data transfers from the underlying protocol and topology that is capable of abstracting high-level memory requests on the address basis. Certain master IP-cores, such as the PE or accelerators, can act as masters on the CA. They issue a request, which could be either read or write, to a particular slave node (IP) such as the main memory or a peripheral. In Figure 2.7, the CA interface is depicted. The read and write operations are handled in the physical address space⁶.

Accelerators and Peripherals

Hardware elements represent the groups of co-processors or slave devices connected to the CA. Typical examples are GPUs, DSP, display subsystems or networking interfaces. The first category includes components providing Direct Memory Access (DMA) capabilities. This means the component acts as a master device on the CA on the one hand, but is also considered a companion device of any logic running on a PE on the other. General purpose co-processors are implemented this way. The second category includes components of which are not capable of accessing the memory on their behalf. They only provide a slave interface to PEs. Typical examples are serial interfaces or a Network Interface Controller (NIC).

⁶Compare Section 2.1.4 for further information on address levels.

Virtualization Extensions

Following the previously given classification of virtualization, the referred to system facilitates a Type-0 virtualization class. In the following, the key implications to facilitate Type-0 hardware virtualization are given [120].

Hypervisor execution mode: The introduction of a new Hypervisor execution mode, of higher priority than supervisor mode in which the system is usually executed. This will enable the VMM to execute at a higher privilege than the guest OSs, and the guest OSs to execute with traditional operating system privileges, removing the need to employ para-virtualization techniques.

Interrupt Provisioning: The provision of mechanisms to aid interrupt handling, with native distinction of interrupt destined to secure monitor, hypervisors, currently active guest OSs or non-currently-active guest OSs.

Multi-Level address management: The provision of a system MMU to aid memory management, supports: multiple translation contexts for multiple DMA capable masters, two levels of address translation and hardware acceleration and abstraction.

Operational Environment

The ToE is considered to instantiate a sub-system of a system, meaning that it resides in a broader functional and technical context. It not only has to fulfil functional and operational requirements of that context but also orthogonal aspects such as quality goals of that given context have to be applied. Technically, it is considered that the ToE is connected to its context using domain-specific, wired or wireless, technologies. In this work, the automotive domain specific use-case of a DCU is taken as an example to instantiate the considered class of ToE. In Chapter 3, a case study including a domain-specific description of the ToE is given.

System States

System states are relevant to the consideration of security of a ToE. The diverse security functions are often only effective in a specific state of the system. Therefore, in order to address particular security needs, it is necessary to differentiate the states. Security relevant states in the life cycle of the ToE focus primarily on situations when security functions are applied to enforce the security objectives. These states of the life cycle are listed in the following:

Startup: The startup or boot phase is crucial for security considerations. In this phase, the chain-of-trust is built up. Most commonly, this is referred to as a secure boot, where the authenticity and integrity of the system are attested and enforced (compare with example [156]).

Operate: During the runtime, the system needs to be kept secure. Since the ToE interacts with its operational environment over defined interfaces, interference induced by adversaries might occur. Therefore, appropriate security functions will take effect for those situations.

Offline: The state when the system is not used. This means it is turned off or it is suspended and no function is activated. Here, the aspect of information disclosure and confidentiality of data resting inside the system are of concern.

Technical Realization

This section describes the particular technique of realizing an AMP system hosting several ADs. The general idea is based on the memory-map resource utilisation paradigm in today's embedded systems. Memory-map paradigm means that every resource in the hardware platform is accessible through a dedicated memory address. This concept is commonly referred to as memory mapped I/O [167]. Hence, the memory addresses are the handles that assign hardware elements to particular logical domains (AD). Accordingly, as a preliminary, it is considered that the ToE applies a central memory-map that represents its hardware elements. The mapping is handled by dedicated integrated functions that control accesses to a particular address. Technically, it is considered that this is implemented by a hardware MMU or hardware Memory Protection Unit (MPU). The central memory-map is considered to realize the configuration of the MMUs or MPUs. In Figure 2.8, the concept of a system memory map is visualized. The figure shows two PEs each of which instantiate an AD. Two peripherals are mapped to the PEs. *Peripheral 1* is mapped to *AD 1* and *Peripheral 2* is mapped to *AD 2*, accordingly. Furthermore, the memory is virtually divided into two partitions. Each of these partitions are mapped to the particular ADs, respectively. As a result, the combination of several addresses which are mapped to the same PE realizes an AD.

Multi Level Address Translation

The ToE is considered to implement multiple address levels. This is necessary to handle multiple domains on the hardware while maintaining the possibility for the

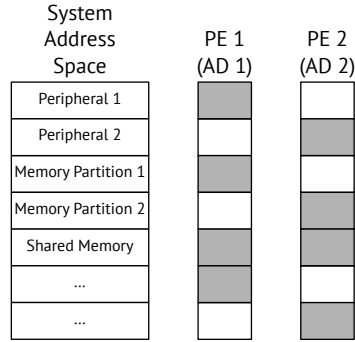


Fig. 2.8 System memory-map concept.

software-stacks to utilize the hardware MMUs for address space separation. In the following, the three considered address levels are defined.

Virtual address: This space is typically maintained by the software-stack. Mainly, the purpose is to provide separate address spaces to each process of an OS, for example. Historically, the introduction of Virtual Addresss (VAs) was also motivated by the lack of enough main memory. VAs memory handling allows swapping memory pages onto persistent memory storages. Usually, the VA differ from the actual physical position within the main memory.

Physical address: Physical Address (PA) represents the address of the physical position in the main memory. The PA covers the entire addressable memory space of a system. Accordingly, the system memory-map is represented by PAs.

Intermediate physical address: Along with the introduction of the virtualization extensions to the PEs⁷, a further level of indirection was introduced to make memory accesses of virtual machine guests manageable. This level is referred to as Intermediate Physical Address (IPA).

As a result of the three address levels, a two-staged memory translation is implemented in the ToE. The translation process goes as follows:

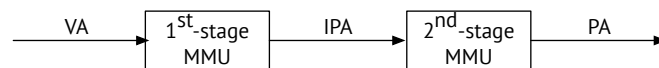


Fig. 2.9 Two-stage memory translation.

⁷Compare Section 2.1.4

MMUs are configured by so-called translation tables. These tables associate an input address to its designated output address. According to the three address space types (VA, PA, IPA), two indirection levels are needed, meaning that, two translation tables are necessary. This principle is depicted in Figure 2.10.

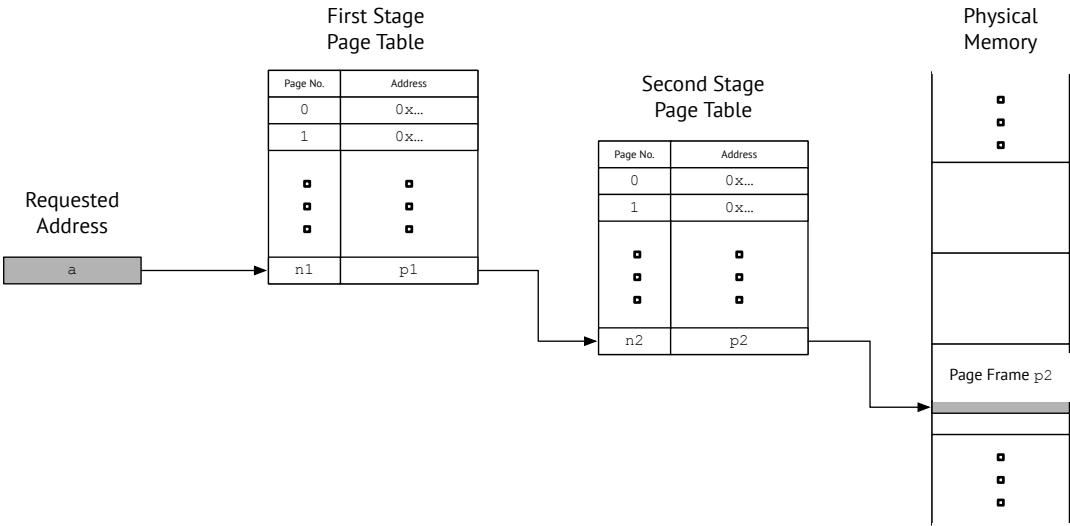


Fig. 2.10 First and second page-table.

2.1.5 Security Problem Definition

Primary Assets

The memory of AD contains valuable assets that are of interest for an adversary. As such, on the intermediate level, the memory partitions are considered. The value depends on what is contained in the partition. This is then represented by the next higher level, for example, the memory layout of the OS on the software-stack of the AD. Furthermore, the MPSoC components which are assigned to a particular AD are crucial assets. In the following, the primary assets are listed.

- Software-stack instructions:** Machine code which is executed by a PE. The particular machine code or assembly code is dependent on the architecture of the PE. The instructions implement the control-flow of a functionality.
- Security properties:** confidentiality, integrity, availability, authenticity, authority

Software-stack data: Data that is processed by the ToE.

Security properties: confidentiality, integrity, availability)

Configuration data: Data that facilitates the configuration of the ToE and the ToE's Target Security Function (TSF).

Security properties: confidentiality, integrity, availability, authenticity

AD resources: Resources that are assigned to and utilized by a particular AD.

Security properties: integrity, availability, authority

Secondary Assets

The ToE is expected to run applications and functions with widespread quality goals. Here, the following quality goals are implied: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, portability [85, p. 4].

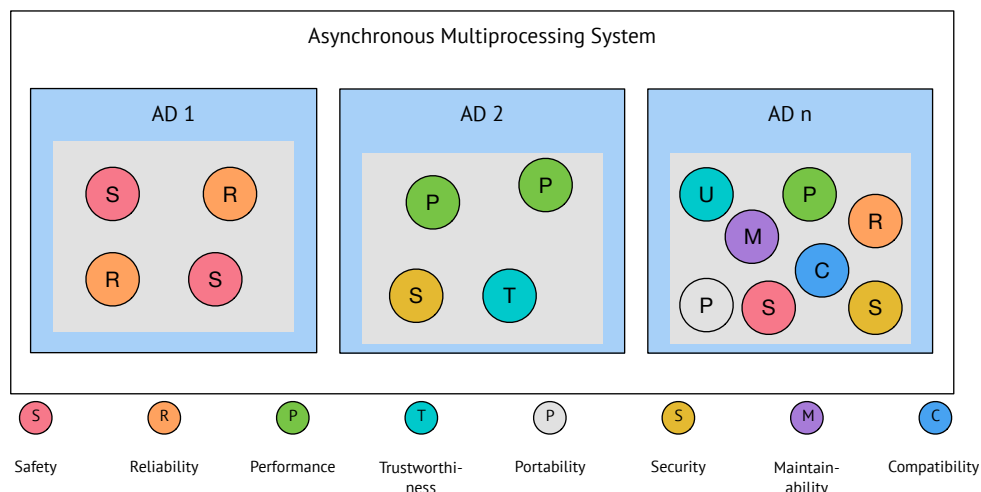


Fig. 2.11 System functions with a focus on particular product qualities.

Threats

It is considered that the previously stated assets are potentially put on risk by the following threats.

Tampering: [TH.Tamper] Modification of the integrity of all assets of an AD.

Information Disclosure: [TH.ID] Disclosing of an AD's information, including for reconnaissance purposes.

Denial-of-Service: [TH.DoS] of an AD

Elevation-of-Privilege: [TH.EoP] of an AD or its subcomponents.

2.1.6 Security Objectives

Integrity For each asset, the ToE must preserve integrity in order to isolate all ADs from each other.

Obfuscation: For each asset, the ToE must preserve the means to limit the impact of compromises in order to mitigate reconnaissance efforts.

Resource availability: For each asset, the ToE must preserve the means to limit the impact of competing accesses to shared resources of particular AD

Resource authorization: For each asset, the ToE must preserve the means to control the access to resources dedicated to a particular AD

Security Mitigation Properties

Types of Security Risk Mitigations The objective regarding the protection of the ToE is to consider the four modes of threat mitigation. These modes are *prevent*, *detect*, *reduce* and *fix*. This relates to the differentiation of Andy et al. in [8]. The authors divide the time axis into four modes of security mitigations:

"Security needs to be designed into the vehicle architecture from the very start and it must furthermore be maintained throughout the vehicle's entire lifecycle. Contrary to common belief, security is much more than prevention only." [8, p. 3]

Furthermore, the aspect of *deterrence* should also be considered. Deterrence takes place before an attack would be mounted by the attacker. It means that a hurdle in the security countermeasure is too high so that an attacker could try to break into the system. As a result, it deters him from attacking and would be, therefore, the ultimate goal of protecting a system.

Evolution Within the ToE Life Cycle The issue of securing the ToE depends on the evolutionary state inside the Product Life Cycle (PLC) or outside within the product-line. Generally, the PLC is mainly focused on when it comes to security processes. PLCs for automotive products are roughly dividable into two main phases.

First, in the *pre-Start of Production (SOP)* phase, the product will be developed using a suitable development cycle such as the v-model. Second, in the *post-SOP* phase, the product needs to be maintained. From a security perspective, two major goals are spread over these two phases. The first goal is to create a state in which the system can be treated as secure. In other words, one needs to be aware of risks in the first place and to mitigate or accept them in the second. The second goal is to keep a certain risk threshold in which the system is still in this secure state. Most commonly, the applied method to find security requirements in the pre-SOP phase is risk assessment [81]. For every function the system has a potential impact and likelihood that will be analysed. Concerning the particular risk, a risk treatment phase follows and the specific strategies to mitigate those risks will be defined. Those mitigations are fed as logical and technical requirements into a system design. In the verification and validation phase in the v-model, appropriate security testing methods are applied to raise confidence in the absence of severe vulnerabilities. Security testing methods include fuzzing (negative testing) and penetration testing. In other words, during the development, the foundations for determining the security requirements so they are accompanied by techniques to gain confidence in the derived logical and technical security architecture. During the post-SOP phase, the system should be observed. An occurring incident will be responded to in an organized manner. Accordingly, if this response requires a security update (software), the development of this update will traverse the secure development v-model for the new function. Limiting factors for the upcoming security mitigations are immutable components of the system which might be functional or technical. A prominent example is the hardware platform which is significantly hard to modify once it is deployed. However, this immutability is not only true for products that are already deployed, as within product lines, the engineering strategy such as top-down and bottom-up might also practically imply further restrictions as well. For example, a hardware platform is to be integrated (bottom-up) for a certain set of software functions. Some of the functions then need to be fitted and developed onto (top-down) the hardware platform. This also implies restrictions for security solutions. This might appear in many reuse situations in automotive product line development. To summarise, security solutions need to fit into this evolving landscape.

2.2 Security Engineering

This section describes the aspects of security engineering that aim for securing an AMP-system. Given the automotive context, it is elaborated upon as to how the engineering of problem space and solution space is handled in security. Whereas the latter deals with defining countermeasures and secure system designs, the former aims at developing the awareness about adversarial behaviour against the ToE, which is the identification of potential threats.

The outcome of this section is the security methodology that has been applied to gain the research results in this work. This methodology, which is elaborated upon in Section 2.2.2, is motivated and aligned with well-established frameworks and methods.

2.2.1 Introduction to Security Engineering

The Role of Security and Secure Systems

In general, implications of security are elaborated upon within this section. These include the goals of security engineering, the correlation with a typical automotive product life-cycle and an outline of activities along within this life-cycle.

It is not trivial to define the meaning of security. First of all, it is necessary to differentiate between the terms *secure* and *security*. Both terms imply a particular perspective to a system or a component of a system. *Secure* means a property of something, such as a system element describes. For example, an AMP system is a secure AMP system if something has been done to make it secure. This might be a procedural, functional/technical or organisational treatment. On the contrary, *security* describes a purpose or a functional justification of something that is in place. For example, a firewall is a security function because it filters unwanted network accesses. To summarize, something is secure when the appropriate security functions are applied. However, a secure system is not the sum of its security functions. It also concerns the design of the particular system. This is where the term *secure-by-design* originates. It means that the way the system is constructed avoids certain unwanted weaknesses. For example, if the availability of a component is a crucial security goal, a reasonable design tactic would be to introduce a redundant component.

Goals of Security Engineering

The overall goal of manufacturers is to deliver a product that meets all stakeholder requirements and wishes. Regarding security, this is to deliver a *secure* product.

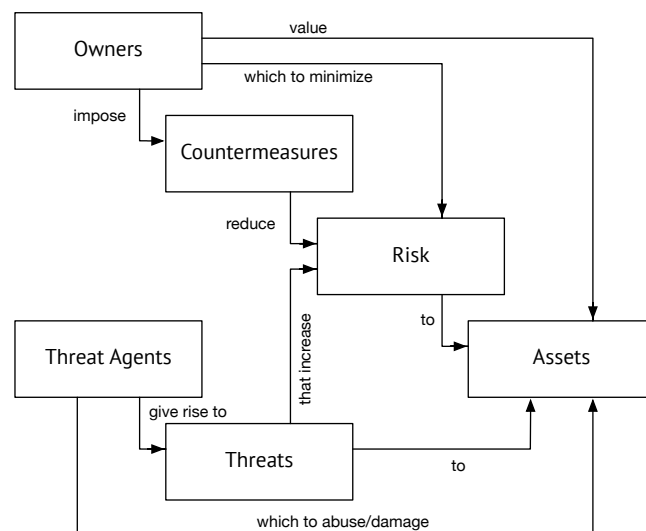


Fig. 2.12 The fundamental dependencies of actors and artefacts in security [41].

Accordingly, all activities in security engineering focus on two aspects. First, at which state is the product secure and second, how to reach and keep that state. Additionally, to maintain and keep the product in a secure state is a challenge that arises from the fact that security engineering is a continuous and evolving activity over the entire lifetime of a product.

To secure a system, it is necessary to find the suitable design or security functions. The question is, what is necessary and suitable? In the field of security, the answer is approached by thinking about what has value to the owners and what risk is expected that the asset is going to be compromised. Accordingly, this can be seen by thinking of two parties competing for an asset. Owners value their assets, and therefore, they wish to protect them from compromise. On the contrary, threat agents wish to abuse and may damage such assets. Accordingly, they put effort into adverse actions and increase the risk of an asset to being attacked. The owners again, impose countermeasures to reduce the risk to assets. The CC methodology introduced these dependencies which are also depicted in Figure 2.12. A proper security engineering framework deals with the dependencies as mentioned above of actors and artefacts.

Engineers that envisage building a secure system encounter three main questions and therefore goals. These goals are stated in the following listing and aligned to three major areas of the CC [176].

Goal 1: Identify security needs

Goal 2: Definition of countermeasures

Goal 3: Assurance of countermeasures

In the following paragraphs, common methods are introduced which address the previously mentioned goals.

Common Criteria The common criteria for information technology security evaluation (in short CC) have been introduced to permit comparability between independent security evaluations [176].

"The CC is useful as a guide for the development, evaluation and/or procurement of IT products with security functionality." [176, p. 11]

The CC are standardised in the ISO15408 standard [84] for certification intents of hardware, firmware and software. Historically, the CC arose from certain methodologies of several governances including the trusted computer system evaluation criteria, the so-called *Orange-Book*, of the United States Government Department of Defense and, furthermore, the counterpart of European governmental institutions. The CC define ToE to describe the system to be evaluated. ToEs will be evaluated on the basis of a particular Protection Profile (PP). In a PP, typical security requirements are applicable to a specific class of devices. PP can serve as a template for a ToE Security Target (ST) which defines the security properties a specific ToE should have. The PP and ST formulate the Security Functional Requirement (SFR) and Security Assurance Requirement (SAR) on which the ToE is finally evaluated against. The intention of SFR is:

"These requirements describe the desired security behaviour expected of a Target of Evaluation (TOE) and are intended to meet the security objectives as stated in a PP or an ST." [175, p. 13].

whereas SAR aims at the

"(...) evaluation criteria for PPs and STs and presents evaluation assurance levels that define the predefined CC scale for rating assurance for TOEs, which is called the Evaluation Assurance Levels (EALs)" [174, p. 13].

An important aspect of the CC are the Evaluation Assurance Levels (EALs). They provide a numerical rating indicating the activities and the depth and rigour of the evaluation within a scale of one to seven. As an example, at EAL 7, the ToE design

must be formally verified and stringently tested. Whereas on EAL level 1, only basic functional tests need to be conducted.

Aside from the functional aspects of a ToE, the CC also include a vulnerability assessment method.

"The purpose of the vulnerability assessment activity is to determine the existence and exploitability of flaws or weaknesses in the TOE in the operational environment." [173, p. 402]

The main factors for conducting a vulnerability assessment are

- the identification of potential vulnerabilities,
- an assessment to determine whether the identified potential vulnerabilities could allow an attacker with the relevant AP to violate the SFRs,
- and penetration testing to determine whether the identified potential vulnerabilities are exploitable in the operational environment of the ToE.

AP plays an important role in the determination and guidance of further activities in the analysis. It indicates system elements of the ToE that imply a certain level of potential to be attacked. The particular AP level that is aimed for depends on the EAL of the ToE. Accordingly, the AP is also used to express a certain amount of resistance. Moreover, and in addition, it is used to define a target potential. This means that countermeasures or SFR can be required to limit the AP to a particular level according to the EAL. In Section 2.2.1, the AP characteristics and dependencies to a security methodology are elaborated upon in more detail. Further risk assessment frameworks also adopt the concept of AP to quantify the likelihood of particular security risks⁸.

As a result, the CC cover the above-mentioned goals (*Goal 2* and *Goal 3*), regarding the structured definition of security needs, countermeasures and the evaluation as such. However, they do not cover all aspects that are necessary to reach the above aforementioned goals of security engineering. This is particularly true for *Goal 1*. The identification of crucial assets is not a concern CC and is considered as being

"(...) outside the scope of the CC, but used as an input into the development of the PP/ST in terms of the Security Problem Definition (...)" [173, p. 417]

⁸These frameworks are elaborated in the following paragraphs.

Risk and Threat-based Security Engineering Frameworks

Risk assessment is the commonly applied method for determining security requirements in development projects within the automotive context. According to Freund, the following questions with regard to risk define the scope of risk based assessments.

"How much risk do we have?

How much risk is associated with ...?

How much less (or more) risk will we have if ...?" [49, p. 1]

A security risk is quantified by relating two major components: impact and likelihood ($Risk = Impact * Likelihood$). Risk is related to an unwanted incident that is anticipated. Impact embodies the expected damage or negative effect to a system's stakeholder/owner. Likelihood quantifies the expected probability of the unwanted incident to occur.

In this paragraph, automotive related risk assessment frameworks are introduced. Furthermore, methods and tools to implement the framework are described. The section concludes with the generalized essentials of security engineering in the automotive context. Additionally, the aspects of integrating research and novel observations into the previously given context are discussed.

SAEJ3061. In January 2016, the SAE International issued the *cybersecurity guidebook for cyber-physical vehicle systems* [81] in a standard for surface vehicles. By issuing this standard, the SAE International is the first institution that has fostered a security process applicable to the automotive environment. As a result, this standard can be interpreted as an important foundation for automotive related security considerations:

"The recommended practice provides guidance on vehicle Cybersecurity and was created based on, and expanded on from, existing practices which are being implemented or reported in industry, government and conference papers. [81, p. 5]

In general, the standard includes a definition of a complete life-cycle process framework with the intent to be tailorable to a company's development process. Furthermore, information on common existing tools and methods for the in-vehicle system development are given[81]. The SAE International aims at providing a foundation for further standards on information security for automotive development processes as well as management frameworks for product life cycles. As a basis, they align every process element to the well-established functional safety process given in the ISO26262 [83]

standard. They state that the two disciplines, security and safety, have some overlap between the engineering activities and similar objectives. Safety aims at defining a state which does not cause harm to life, property, or the environment, whereas information security aims not allowing the exploitation of vulnerabilities which would lead to losses such as financial, operational, privacy, or safety. The standard elaborates every element in a common v-cycle development process in detail. Additionally, the guidebook considers the management and supporting processes as well.

EVITA. The European commission for information society and media and industrial partners funded the project E-Safety Vehicle Intrusion Protected Applications (EVITA) [66]:

"The objective of EVITA is to design, verify, and prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering, and sensitive data are protected against compromise. [66, p. 1]"

This approach includes methods for security requirements analysis, secure on-board architecture design, implementation and prototype-based demonstration. Briefly spoken, the process aims at identifying security goals on the system feature level. Therefore, it identifies these features of the system and assigns potential threats to it. This is adopted from the Hazard and Risk Assessment (HARA)⁹ and is called Threat and Operability Analysis (THROP). The impact component of risks is approached by anticipating the expected damage to four aspects. These aspects include safety, finance, operation and privacy. The likelihood component of the risk is approached by the AP quantification according to the CC.

The HEAVENS Project. The HEALing Vulnerabilities to ENhance Software Security and Safety (HEAVENS) security model [81] focuses on methods, processes and tool support for threat analysis and risk assessment concerning the vehicle E/E system. Mostly the model follows the EVITA approach in defining the impact severity, including some enhancements in certain aspects. As an example, the authors give reference to commonly accepted methods by rating impacts to the four security objectives (safety, operational, financial and privacy). For example, the estimation of safety impacts will be supported by methods adopted by the ISO26262 standard [83]. Moreover, the risk likelihood is determined by the CC's AP. Additionally, HEAVENS adopts Microsoft's

⁹Safety related risk assessment [83]

Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, Elevation of Privilege (STRIDE) threat model to determine potential threats in a structured manner.

Factor Analysis of Information Risk. Factor Analysis of Information Risk (FAIR) is a methodology for measuring and managing information risk [49]. FAIR provides a practical and credible framework to understand, measure and analyse security risks of any size or complexity. The framework provides an ontology to decompose risk into the

"(...) probable frequency and probable magnitude of future loss." [49, p.27]

In Figure 2.13, the main factors of risk determination in FAIR are shown. One of the key aspects of the FAIR approach is to quantify occurrences by rating probabilities such as the probability that a threat agent's action will result in a loss.

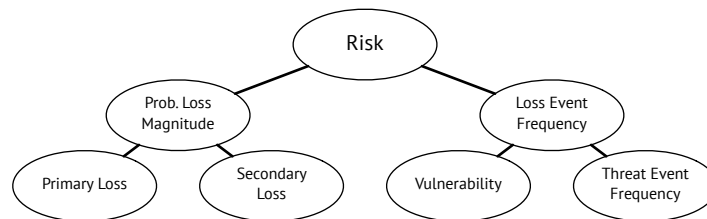


Fig. 2.13 FIRST ontology. (adopted from [49, p. 31])

Methods and Tools

Risk Quantification.

Impact Severity. In the following the impact severity factors are proposed. The factors refer to the context of automotive risk assessment [66].

Safety: Ensure safety of the vehicle occupants and road users.

Financial: Prevent negative impact to the financial situation of a concerned party.

Operation: Maintain operational performance of all vehicle functions.

Privacy: Protect privacy of drivers or the intellectual property of the manufacturer.

To each of those threats, a severity of four security objective categories will be assigned (compare with Table 2.1).

Table 2.1 Impact severity classes [67].

Severity Class	Safety	Financial	Operational	Privacy
0	no injuries	no financial loss	no impact to operational performance	no unauthorized access to data
1	light or moderate injuries	low-level loss ($\approx 10\text{€}$)	impact not discernible to driver	anonymous data only (no specific vehicle driver data)
2	severe injuries (survival probable): light moderate injuries for multiple vehicles	moderate loss ($\approx 100\text{€}$), low losses over multiple vehicles	driver aware of performance degradation; indiscernible impacts for multiple vehicles	identification of vehicle or driver, anonymous data for multiple vehicles
3	life threatening (survival uncertain) or fatal injuries	heavy losses ($\approx 1000\text{€}$), moderate losses over multiple vehicles	significant impact on performance, noticeable impact for multiple vehicles	driver or vehicle tracking, identification of driver or vehicle for multiple vehicles
4	life threatening or fatal injuries for multiple vehicles	heavy losses over multiple vehicles	significant impact for multiple vehicles	driver or vehicle tracking for multiple vehicles

Attack Potential. The potential of an attack will be determined. The resulting AP quantifies the probability of a potential attack. To determine the probability, five characteristics will be factored. These characteristics include elapsed time, expertise, knowledge of the system, window of opportunity, equipment required. The respective qualitative rating scale is shown in Table 2.2.

The risk components have values assigned which will be summarised with the resulting AP factor. In Table 2.3, the AP factors are mapped to a severity level as well as an attack probability. The attack potential levels reach from *Basic* to *Beyond High* and the corresponding likelihood from 5 to 1, respectively.

Table 2.2 Attack Potential rating table [173].

Factor	Level	Value
Elapsed time	\leq Day	0
	\leq Week	1
	\leq 1 Month	4
	\leq 3 Months	10
	\leq 6 Months	17
	$>$ 6 Months	19
	Not practical	∞
Expertise	Layman	0
	Proficient	3
	Expert	6
	Multiple experts	8
Knowledge of system	Public	0
	Restricted	3
	Sensitive	7
	Critical	11
Window of opportunity	Unnecessary/unlimited	0
	Easy	1
	Moderate	4
	Difficult	10
	None	∞
Equipment	Standard	0
	Specialized	4
	Bespoke	7
	Multiple bespoke	9

Correlation of Risk Factors. The combination of the components impact severity and attack likelihood is shown in Table 2.4. According to the CC's EAL levels the risk levels reach from 0 to 7+. Impact severity classes are weighted differently in the given table. Impacts to safety (denoted by S) independently modify the severity class. The other impact factors (financial, operational, privacy) are denoted by C and are represented by the maximum value out of these three factors.

Definition of Required and Residual AP. In the CC evaluation guidelines for vulnerability assessment, it is proposed to use the AP as an assurance criterion. The AP is not only used to express a likelihood, but can also be used to state a targeted capability to resist attacks. In other words, it can be used to describe a required

Table 2.3 Attack Potential level and likelihood [173].

Range	Required attack potential	Attack likelihood
0 - 9	Basic	5
10 - 13	Enhanced basic	4
14 - 19	Moderate	3
20 - 24	High	2
> 25	Beyond high	1

quality of countermeasures. Evaluations for whether or not proposed risk treatments are sufficient in the given context are possible. Since the risk cannot be eliminated, a required residual AP can be attached to the risk acceptance level.

In Table 2.5, the risk levels are correlated to required AP and the corresponding residual AP level. This correlation is based on the evaluation assurance components for vulnerability assessment (compare [174, p. 32])) and the vulnerability testing and AP given in [173, p. 418]). The latter assigns a particular component (*AVN.X*) to the required and residual AP, whereas, the former relates EAL to the assurance components. In the following table, this arithmetic is adopted to correlate a risk level with the targeted AP.

Threat Modelling. Threat modelling is a method to show what can go wrong concerning something that has to be built. Shostack formulates threat modelling as follows:

"Threat modelling is the key to a focused defence. Without threat models, you can never stop playing whack-a-mole. In short, threat modelling is the use of abstractions to aid in thinking about risks." [154, p. XXIII]

To make use of threat modelling, two things are necessary: first, a model of threat types that covers all possible types of hazards to a system element and second, a suitable system model to apply the previously mentioned threats. In research and industry, a plethora of threat models have been introduced [42]. A widespread and well accepted threat model is Microsoft's STRIDE approach. STRIDE is introduced in the following paragraph.

STRIDE. STRIDE is a mnemonic for things that go wrong in security [154]. Spoofing, tampering, repudiation, information disclosure, denial-of-service and elevation of privilege are threat types which Microsoft has introduced along with their

Table 2.4 Security risk level [67].

Impact Severity		Attack likelihood				
		AL = 1	AL = 2	AL = 3	AL = 4	AL = 5
C=1	S = 1	0	0	1	2	3
	S = 2	0	1	2	3	4
	S = 3	1	2	3	4	5
	S = 4	2	3	4	5	6
C=2	S = 1	0	1	2	3	4
	S = 2	1	2	3	4	5
	S = 3	2	3	4	5	6
	S = 4	3	4	5	6	7
C=3	S = 1	1	2	3	4	5
	S = 2	2	3	4	5	6
	S = 3	3	4	5	6	7
	S = 4	4	5	6	7	7+
C=4	S = 1	2	3	4	5	6
	S = 2	3	4	5	6	7
	S = 3	4	5	6	7	7+
	S = 4	5	5	6	7+	7+

comprehensive security development life-cycle [68]. In Table 2.6, the threat types are associated with the corresponding security properties. Furthermore, a short description provides the meaning of the threats.

Data Flow Diagrams. DFD are suitable models to apply threat models such as STRIDE:

"DFD's often form an important role in the design of information systems. Their intention is to model the process aspects of an information system (...). They are often used in the first phases of information analysis to establish a global model of an information system which can be further refined." [20, p. 1]

DFD model processes, data storages, data flows and external actors such as shown in Table 2.7.

Threat Model Application. Given a certain system model such as DFD, the threats need to be associated with each other. According to Shostack [154], there are

Table 2.5 Required and Residual AP (based on [173, p. 418]).

Risk Level	Required Resistance AP	Residual AP
≥ 6	High	Beyond High
≥ 5	Moderate	High
≥ 4	Enhanced Basic	Moderate
$\geq 2\ 3$	Basic	Enhanced Basic
1	Basic	Enhanced Basic

Table 2.6 Microsoft STRIDE Threats. [154]

Threat	Property Violated	Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself.
Tampering	Integrity	Modifying something on disk, on a network, or in memory.
Repudiation	Non-Repudiation	Claiming that you didn't do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?
Information Disclosure	Confidentiality	Providing information to someone not authorized to see it.
Denial-of-Service	Availability	Absorbing resources needed to provide service.
Elevation of Privilege	Authorization	Allowing someone to do something they're not authorized to do.

two general methods to associate threats with model elements. One is the STRIDE-by-element, and the other is STRIDE-by-interaction. As the names may imply, the first method seeks to associate each element in the model with a defined set of threats. The second method assigns the threats to a pair of elements which are related with in data flow. In [18] the authors compared both methods and concluded that STRIDE-per-element is more efficient. This section elaborates on the application of STRIDE threats to the DFD elements (STRIDE-per-Element). Not every STRIDE threat applies to all DFD elements, due to their nature. For instance, a process can be spoofed because it logically represents an instance for something that can be impersonated. Data flows, as they are, on the contrary, represent just a logical connection which can not be impersonated. However, in the case that two processes are connected by a data flow,

Table 2.7 Data flow diagram

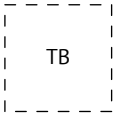



Element	Description
	Trust boundaries surround a certain set of nodes which belong logically or physically together and set a border for untrusted elements in the system.
	In data storages, data are stored and represented as files, memory, databases, etc..
	Processes (such as $P1$) are nodes which handle and compute information.
	Representing data flow across <i>object</i> from process $P1$ to process $P2$.

Table 2.8 Threats per DFD Element.

DFD Element	Threat					
	S	T	R	I	D	E
Process	✓	✓	✓	✓	✓	✓
Data Flow		✓		✓	✓	
Storage		✓		✓	✓	

the spoofing threat (impersonation) then applies for both processes. For the process that is sending data, wants to ensure to send it to an authentic entity. Accordingly, the receiving process might want to ensure that the data was sent by an authentic origin. Table 2.8 shows the STRIDE to DFD element correlation.

Attack Modelling by Attack Trees. Attack trees are a formal method to show the security of a system. Basically, potential attacks will be visualised in a tree structure [148]. They support a security analyst in understanding attack goals, motivations and causal interconnections of certain intermediate steps an attacker may take to achieve his main goal. In the EVITA approach [67] to attack trees, a generic structure is proposed consisting of the following levels:

Level 0: attack goal (analogous to the top event in a fault tree)

Level 1: attack objectives

Level 2: attack methods

Level 3: $(n - 1)$: intermediate goals / methods

Level n : asset attacks (the base level methods of performing an attack; analogous to base events in a fault tree).

Penetration Testing. Penetration testing (pentesting) is an activity which aims at examining vulnerabilities of a ToE during the evaluation. In many contexts this is referred to as negative testing.

In a sophisticated analysis setup, the results from the risk assessment can be applied to direct a pentest to the riskiest system elements. The risk assessment then will provide hypothesised threats. Notably, the AP indicates needs for further investigations.

Factorization of Vulnerabilities

Whereas the AP quantifies an estimation of a likelihood for a vulnerability to be exploited, the factorization of vulnerabilities aims at the quantification of a discovered vulnerability.

The leading question is: what does an attacker need to do in order to exploit a particular vulnerability? Hence, the exploitability aspect is defined as such:

Definition 2.1 *Exploitability is the level of difficulty an attacker must overcome [49].*

Accordingly, exploitability expresses what effort is necessary to reach a particular attack goal. In contrast to that, the *threat capability* aims for the given circumstances of an assumed threat agent. So, it answers the question about "What is the attacker capable of doing?". The threat capability is defined as such:

Definition 2.1 *The Threat Capability is the level of force an attacker can apply [49].*

It depends on the particular security engineering framework if the meaning of threat capability or exploitability will be applied. For example, the strength of cryptographic methods is often defined by assuming a certain capability of an adversary. So, the threat capability applies here. In risk-based engineering frameworks the requirements are derived to raise the exploitability to an acceptable amount.

In this work, the exploitability is applied as a metric to evaluate the findings. The threat capability is only applied for informal purposes.

Exploitability Factorization. With the CVSS, an open framework to communicate characteristics of IT vulnerabilities has been introduced. As a standard quantification model, the CVSS aims for an industrial, organisational and governmental application. Particularly, the CVSS is adopted by the international telecommunication union [87] and is used in the National Vulnerability Database (NVD) [127] by the National Institute of Standards and Technology (NIST) to characterize security flaws within their security content automation protocol specification [32].

CVSS Metrics - Base Score. The metrics to score vulnerabilities are categorised as exploitability and impact. These categories describe particular preliminaries, complexity, scope and security properties to break a component [97, 61]. Each category assigns a characteristic, for example, none, low, high. These characteristics correspond to a quantitative value, which represents the severity of the metric. In the end, all ratings will result in a vulnerability score¹⁰.

¹⁰Please refer to [47, p. 18] to retrieve the corresponding equations to calculate the particular scores.

Table 2.9 CVSS Base Score [47].

Score	Description	Values
Entry Attack Vector	This metric reflects the context by which vulnerability exploitation is possible. This metric value (and consequently the base score) will be larger the more remote (logically, and physically) an attacker can be in order to exploit the vulnerable component. The assumption is that the number of potential attackers for a vulnerability that could be exploited from across the Internet is larger than the number of potential attackers that could exploit a vulnerability requiring physical access to a device, and therefore warrants a greater score.	High/ Low
Attack Com- plexity	This metric describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. As described below, such conditions may require the collection of more information about the target, the presence of certain system configuration settings, or computational exceptions. Importantly, the assessment of this metric excludes any requirements for user interaction in order to exploit the vulnerability (such conditions are captured in the User Interaction metric). This metric value is largest for the least complex attacks.	High/ Low/ None
Privileges Re- quired	This metric describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. This metric is greatest if no privileges are required.	High/ Low/ None
User Interac- tion	This metric captures the requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component. This metric determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user (or user-initiated process) must participate in some manner. This metric value is greatest when no user interaction is required.	None / Re- quired

Table 2.10 CVSS Base Score continued [47].

Scope	Formally, Scope refers to the collection of privileges defined by a computing authority (e.g. an application, an operating system, or a sandbox environment) when granting access to computing resources (e.g. files, CPU, memory, etc). These privileges are assigned based on some method of identification and authorization. In some cases, the authorization may be simple or loosely controlled based upon predefined rules or standards. For example, in the case of Ethernet traffic sent to a network switch, the switch accepts traffic that arrives on its ports and is an authority that controls the traffic flow to other switch ports.	Un- changed / Changed
Impact (CIA)	The Impact metrics refer to the properties of the impacted component. Whether a successfully exploited vulnerability affects one or more components, the impact metrics are scored according to the component that suffers the worst outcome that is most directly and predictably associated with a successful attack. That is, analysts should constrain impacts to a reasonable, final outcome which they are confident an attacker is able to achieve.	High / Low / None
-		

Environmental Score The environmental score provides an opportunity to connect risk assessment results with the CVSS in order to modify the particular base scores according to contextual circumstances. The modified base metrics affect all metrics of the base score shown in Table 2.9.

Temporal Metrics Temporal metrics are an opportunity to modify the overall CVSS score due to the current exploit situation.

"The Temporal metrics measure the current state of exploit techniques or code availability, the existence of any patches or workarounds, or the confidence that one has in the description of a vulnerability." [47, p. 12]

The metrics include the *exploit code maturity*, the *remediation level* and the *report confidence*. The maturity of exploit code factorizes the likelihood of the vulnerability

of being attacked. If the exploit code is available in a, for example, functional manner then potential attackers can directly reuse it. In contrast to the exploit maturity, the remediation level factorizes the availability of security fixes in relationship to the discovered vulnerability. Furthermore, the report confidence measures the degree of confidence in the existence of the vulnerability and the credibility of the known technical details [47, p. 13].

Sources for Attack and Vulnerability Analysis

For the analysis and evaluation the consultation of publicly available sources is common practice and required by the CC [173]. In the following, legitimate sources are discussed.

Attack and Vulnerability Databases The MITRE corporation issued two systems to describe and enumerate common security weaknesses and vulnerabilities. In publicly accessible databases such as the NVD¹¹, comprehensive lists of the previously mentioned vulnerabilities are shown.

Well known examples of common enumeration and categorization schemes are the Common Weakness Enumeration (CWE), the Common Vulnerability Evaluation (CVE) and the Common Attack Pattern Enumeration and Classification (CAPEC). Whereas the latter, CAPEC, aims at providing a comprehensive dictionary and classification taxonomy of attack patterns¹² the former schemes, CWE and CVE, enumerate and list weaknesses and vulnerabilities in common software architectures.

Scientific Sources In the domain of public accessible sources for attacks the category of research results plays an important role. Legitimate sources can be university libraries, scientific indexes such as IEEE, ACM, Springer or comprehensive internet databases such as Google Scholar.

Relationship of Security Metrics

In the previous sections and paragraphs particular metrics are introduced, that are applied in particular stages of the security assessment methodology. In Figure 2.14, the relationship between the metrics is depicted.

¹¹<https://nvd.nist.gov/>

¹²<https://capec.mitre.org/>

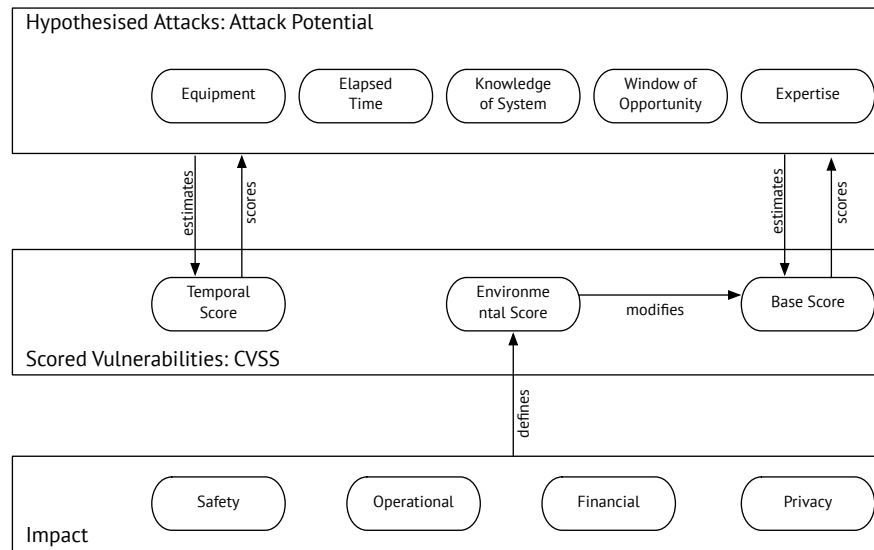


Fig. 2.14 Relationship of security metrics.

2.2.2 Security Assessment Methodology

In this section, the applied security methodology is described. The methodology consists of two major activities: first, the identification and definition of security requirements and second, the evaluation of the proposed countermeasures.

Risk-based Security Requirements Engineering Process

To conclude with the previously given security engineering approaches, here, a generalized approach to conduct security requirements is outlined to serve as a reference throughout this thesis. Security Engineering seeks to cope with the complexity of determining a certain risk on an asset. The risk will guide to the crucial system elements and point out where risk treatments need to be applied. The identification of assets in the system relies upon a suitable threat identification and assessment. The goal, in the end, is to define risk treatments. These are represented as security countermeasures that are required to be implemented in order to mitigate the risk. As a result, the general process components are the risk assessment, vulnerability assessment and risk treatment.

ToE- and Context Profiling: At the very beginning, the ToE must be defined. Here, the functional and technical aspects are elaborated upon. In order to prepare the analysis in the next step, the ToE is decomposed into system elements

(assets) in a granularity that is intended to investigate. Furthermore, an analysis of the threat context is conducted. This defines the attack surface from the ToE environment. Assets in the ToE and potential goals and motivations of adversaries.

Threat and Attack Analysis: In the first analysis step (*Threat Identification*), the identification of assets is aimed in order to identify how they are threatened. An asset is an element that has a specific value to the owner¹³. Threats are considered to categorise how a certain system element is attackable. In this particular case, the STRIDE threat model is applied. Through the application of this threat model¹⁴, each system element is exposed to each of the six threats. This step will deliver a fully covered and exhaustive list of threats associated with the technical system elements. Whereas the threat list is gained deductively, the *Attack Analysis* is characterised as an inductive consideration. The *Attack Analysis* seeks to identify attacks that facilitate the previously identified threats. Attacks are modelled in an attack tree.

Risk Analysis: In the *Risk Analysis*, the two risk components, impact and likelihood, are investigated. The former is achieved by taking the attacks to the system elements (assets) into account. Impacts are quantified by expressing the expected damage to the system owners or other stakeholders. This damage is categorised into safety, operational, financial and privacy and refers to the method given in Section 2.2.1. The likelihood of an attack to occur is assessed by anticipating the minimum effort to mount a given attack. Here, the concept of AP according to the CC is adopted. Taking such information into account is required by the CC assessment in order to comply with high EAL. Compare *Action AVA_VAN.1.2E* in the CC evaluation methodology [173]. The result of the risk assessment is a list of risks with an associated risk level. This works considers two attacks/vulnerabilities in detail, therefore these *Hypothesised Attacks* will be further investigated in the *vulnerability assessment*.

Vulnerability Assessment: The *Vulnerability Assessment* aims at identifying and score vulnerabilities of the ToE. Vulnerability analysis is the structured decomposition of the assessed *Hypothesised Attack*. The outcome of the vulnerability analysis are *hypothesised vulnerabilities* that could be investigated in a Penetration Test. An associated vulnerability score is used as the basis for

¹³As it is described in the security principle in Section 2.2.1

¹⁴compare Section 2.2.1

the effectiveness evaluation in the security evaluation process. Furthermore, the vulnerability analysis states a *security problem* to be mitigated by a risk treatment. In order to create evidence of the existence and practical implications, a penetration test is applied. The penetration test is scored by the *Temporal Score*. The outcome of the *vulnerability analysis* and the evidence of the *penetration test* creates evidence for the *attack potential analysis* of the *risk assessment*.

Risk Treatment: The selection of proper risk treatments comprises the last step in the engineering process. Usual strategies are the avoidance, transfer, acceptance and mitigation of risk. When it comes to the risk mitigation, technical measures are required. These countermeasures have a direct connection with the associated threat category. As an example, if tampering with a system element is identified as a critical risk, the risk treatment to mitigate tampering would be the integrity protection. The appropriate strategy will be defined in the first step. Afterwards, the *Mitigation Concepts* will be analyzed and comprises the actual security solution. The targeted effectiveness quality of the treatments are defined in parallel and evaluated in the security evaluation process.

The reference process shown here demonstrates how to apply available concepts to approach security engineering. However, there are limitations by applying a concept such as the one previously shown. As of today, there is no guidance on defining accepted risk levels. It has to be decided in the context of a particular development project which risk level is acceptable and which one has to be mitigated. Furthermore, the countermeasures to mitigate risks are not defined. The above-given examples and relationship to the threat model is a proposal to encounter the issue, by categorising the countermeasures in a structured manner.

Security Evaluation Process

The aim of the evaluation is to identify the residual risk and the effectiveness of the proposed countermeasures. Intentionally to conduct the evaluation, the previously mentioned process is partially re-assessed. This applies to the process activities which evaluate and score the AP and the vulnerability score. During the risk assessment, the impact analysis is not re-executed. This is motivated by the fact that the actual impact to the functionality does not change if the capabilities of the functionality do not change. The applied evaluation process is depicted in Figure 2.16.

Effectiveness Assessment: In the *Effectiveness Assessment* swim lane, the *Effectiveness Analysis* outputs the modified vulnerability score which is applied

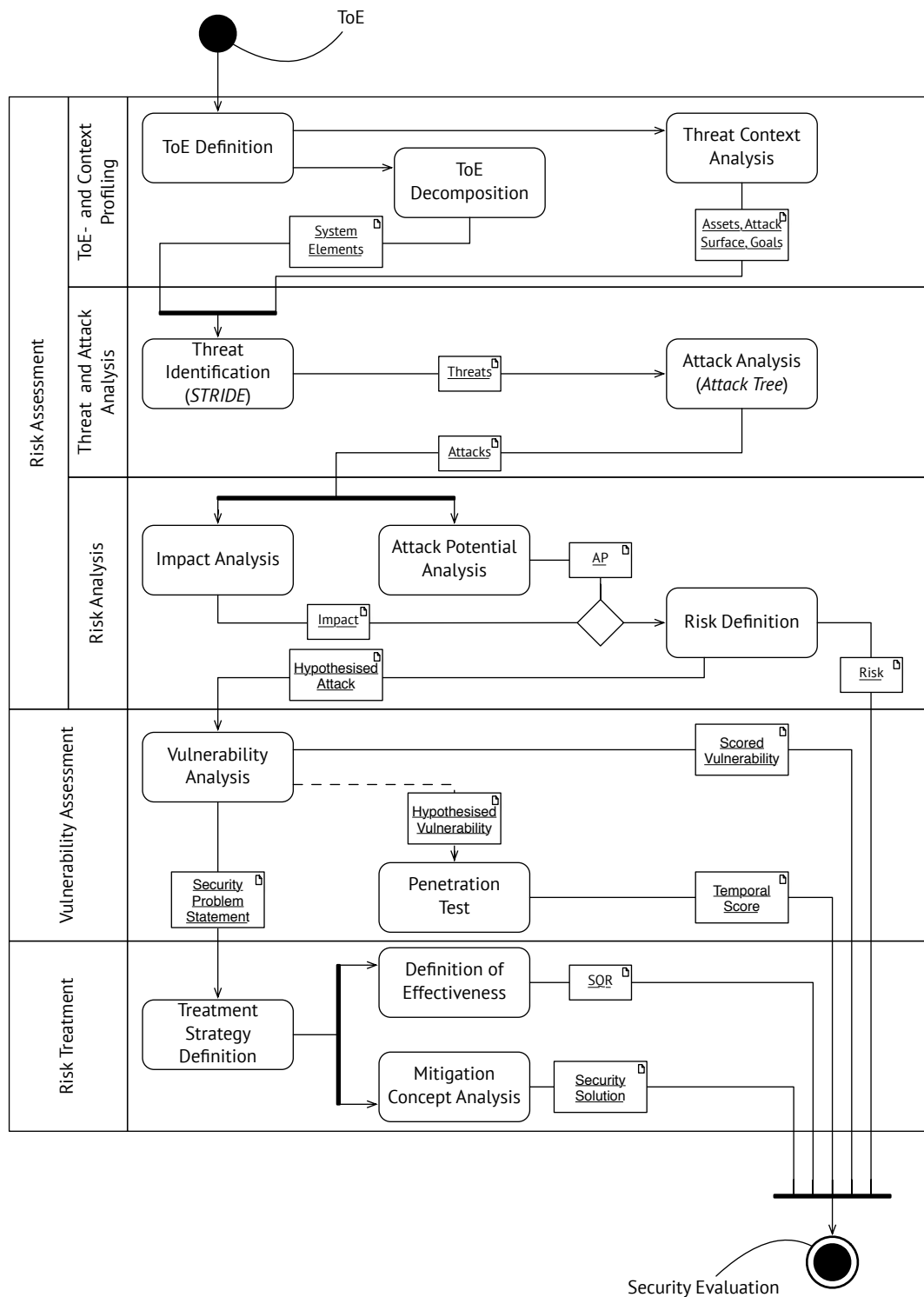


Fig. 2.15 Risk-based Security Assessment Process.

to the ToE with the applied countermeasure. In the *Effectiveness Evaluation* the *Effectiveness Delta* and *Environmental Score* is determined.

Residual Risk Analysis: During the *Residual Risk Analysis*, the *Risk* from the initial *Risk Definition* and the modified vulnerability score is taken into the *Attack Potential Analysis* to analyse the modified AP. The delta of the initial risk (*Risk*) and the residual risk (*Risk'*) indicates the overall risk mitigation-effectiveness of the given countermeasure approaches.

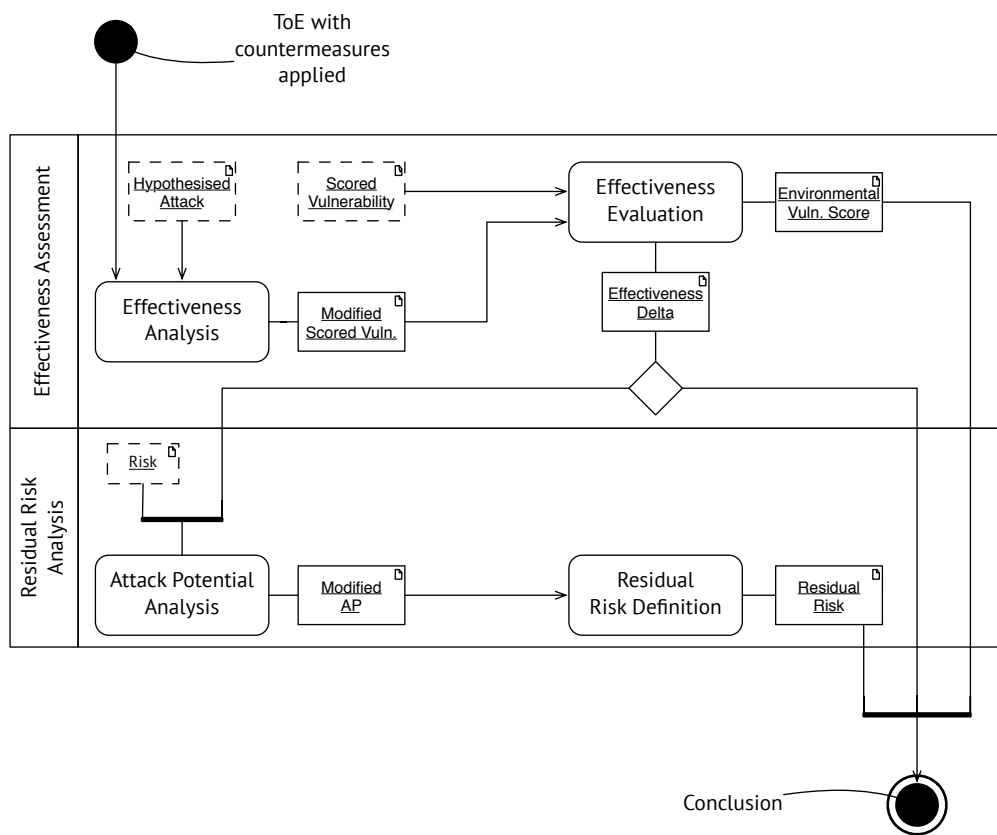


Fig. 2.16 Applied security evaluation process.

Discovering Security Knowledge - Security Cycle

This section seeks to elaborate upon the interplay between the knowledge of security issues and the discovery of new, currently unknown, security issues. More precisely, how does a discovery affect the knowledge base which is taken into account for secure systems?

Two main stages to describe this cycle of security are considered. The first stage is the security *knowledge base* which is comprised of known threats, attacks, vulnerabilities, patterns, etc. The second stage is the security *discovery* or the discoveries which describe a novel threats, attacks, etc. Discoveries shall enrich or enhance the knowledge base. Whereas the knowledge base might evoke new discoveries. These interactions are visualized in Figure 2.17.

Methods to conduct analysis on the knowledge base are of a *deductive* nature. For example, in Section 2.2.2, the risk assessment process is applied. This includes the application of a threat model that represents generic categories of attacks and an attack analysis that should reveal specific instances of the generic threats concerning the specific ToE architecture.

The methods which are applied in the discovery stage are of an *inductive* nature. This means, that a certain ToE is already in the stage to be *specific* and discovered security issues have to be *patterned* in order to enhance the knowledge base. The problem in this kind of inductive analysis is finding the suitable level of abstraction of the discoveries in order to make them reusable. This is caused by the fact that, contrary to general purpose computing, the area of embedded systems engineering aims at creating a system which fulfills a special purpose. This includes all engineering disciplines including, electrical, mechanical, hardware and software engineering. This is particularly the case for the herein considered AMP-based systems. The challenge is to reveal the intrinsic security issue concerning the intrinsic nature of the ToE.

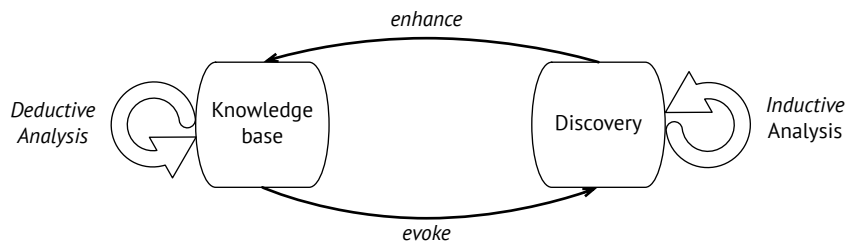


Fig. 2.17 Discovering security knowledge.

2.3 Research Methodology

This section elaborates on how the research is carried out by this work. Hence, first, the research questions are discussed in 2.3. In 2.3 the research artefacts are also given. Lastly, in 2.3 the applied scientific methods are discussed.

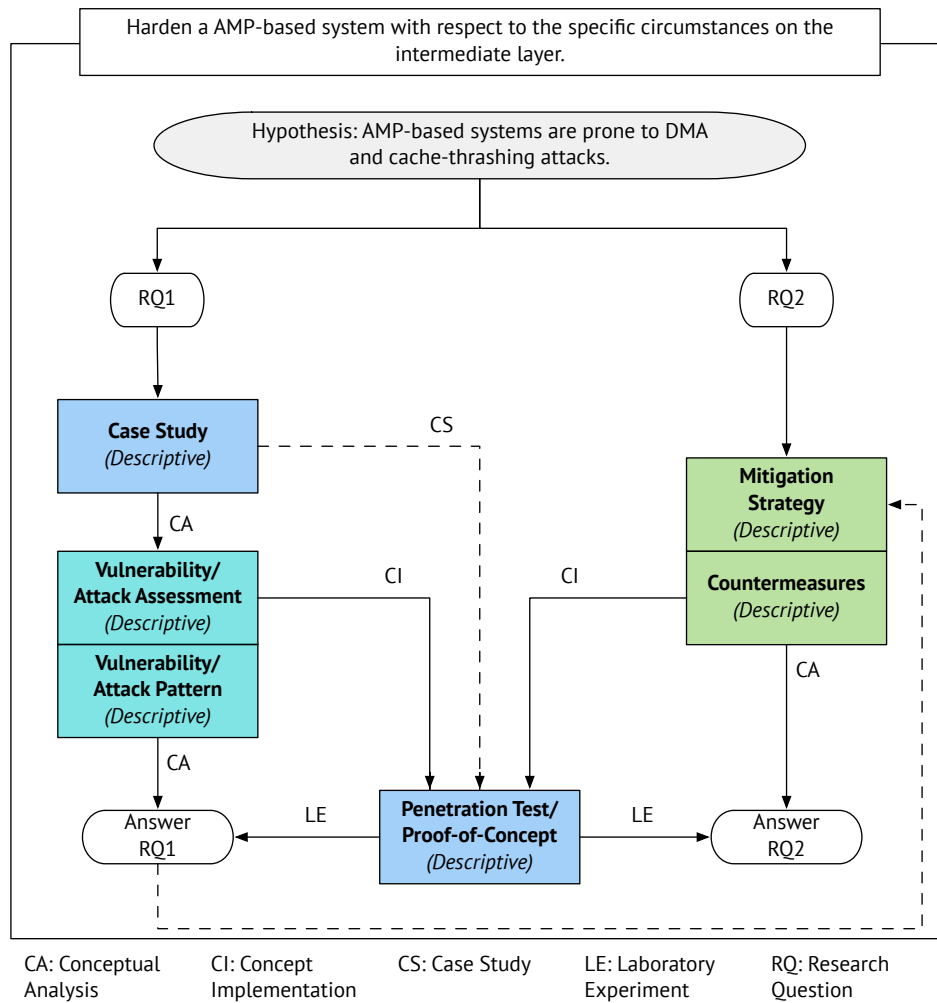


Fig. 2.18 Research Methodology.

Research Questions

This work considers particular security aspects concerning technical constraints of the intermediate layer. Those constraints are implied by a specific construction paradigm for computing systems, which is, in this case, AMP. In general, the research in this work aims for the following:

Research Aim: *Harden an AMP-based system with respect to the specific circumstances on the intermediate layer.*

According to Shaw [153], this type of research is considered a:

"Method or means of development." [153, p. 5]

It, therefore, aims at finding a better way or improve the security of AMP-systems.

According to the research methodology visualized in Figure 2.18, the approach of inductive reasoning is adopted throughout this thesis.

"[...] inductive reasoning [...] is the process by which generalizations are made based on individual instances." [23, p. 27]

It assumes that an AMP-system as it is, is preliminarily characterized in this chapter. The starting point for this research is the following presupposition:

Hypothesis: AMP-based systems are prone to DMA and cache-thrashing attacks.

From this point, the research is conducted by questioning two main areas. The first research question (RQ1) considers the security problem space of the aforementioned presupposed attacks. The other considers the security solution space by dealing with the mitigation of the discovered attacks (RQ2).

Research Question 1 (RQ1): *What is the pattern of DMA and cache-thrashing attacks reflecting the particular AMP characteristics?*

This question is approached by analysing the concepts of DMA and cache-thrashing attacks in order to apply them to an AMP system. These analyses are, furthermore, conceptually implemented in a Proof-of-Concept (PoC) in order to make conclusions about the presupposed hypothesis. The perception of these attacks builds the basis for discovering patterns within the findings. This then evokes the question: *are those findings a general problem of AMP-systems?* The foundation for this endeavour is conducted by a case study of a driver information system. The study establishes the context of the ToE and serves as a running example throughout the thesis. The purpose of this research question is to identify gaps in a protection architecture in a layered system design. This will help to reveal structural weaknesses to define protective means to fulfil security goals or to limit the effects of exploitation.

With respect to the first research question, the second research question focuses on the possibilities to harden the system against the identified gaps in the protection architecture. Therefore, the research question is formulated as follows:

Research Question 2 (RQ2): *How can one mitigate the risk of exploitation of the previously identified vulnerabilities?*

This question is approached by taking the results of RQ1 into account and proposes strategy specific countermeasures based on general mitigation. An important aspect for formulation is the relationship between the identified vulnerabilities and the proposed mitigation approaches.

The purpose of this research question is either to find technical solutions to the previously identified security issues or to find a general mitigation strategy which can be taken into account in the design of AMP systems.

Research Approaches and Methods

In order to find answers to the given research questions, multiple research techniques have been applied. In general, the nature of the approaches are descriptive, formulative and evaluative by following the definition given in [54].

The targeted research combines several disciplines of computer science and security engineering respectively. These include systems engineering, software engineering and security engineering. Accordingly, the research methods are aligned to that areas.

Hence, with respect to [53, 54], the following research methods have been applied:

CA: Gain knowledge by analysing a phenomenon in a structured way [53].

Conceptual Analysis Mathematical: A conceptual analysis is performed by utilizing mathematical techniques. [53]

Concept Implementation (CImpl): Realization of a concept in order to prove a certain degree of feasibility [53].

Case Stuy (CS): A case study investigates a contemporary phenomenon (...) in its real-world context [190].

Laboratory Experiment (LE): Drawing conclusions by an experiment, for example, by collecting and evaluating data [53].

Research Artefacts

According to Figure 2.18 and the research questions given in the previous section, this work includes several types of research artefacts. Here, these are outlined including their dependency to other artefacts.

Case Study: Driver Information System This is approached by providing a real-world instance of an AMP-system. Furthermore, this sets the context for the security engineering methodology. The study, presented in Section 3.1.1, describes a Mixed-Criticality System (MCS) in an automotive context. According to Shaw et al. [153], it is represented as a descriptive model. The input for this artefact is the profile of AMP-systems¹⁵ and the security assessment methodology¹⁶. In the second part of the case study, the risks of the MCS are assessed. This includes the definition of *impacts* and the AP. Based upon the *risk analysis*, the *risk treatment strategy* is defined. A further output of this artefact are *hypothesised attacks* which are assessed in the vulnerability assessment.

Vulnerability Assessment The analysis seeks to endeavour, conceptually, instances of DMA and cache-thrashing in an AMP-system (given in Section 4.1 and 4.2). According to Shaw et al. [153] this is represented as a descriptive model. The inputs of this artefact are the *hypothesised attacks* from the risk assessment of the case study. The AMP-architectural vulnerabilities are evaluated according to the vulnerability factorization¹⁷. The generated outputs are represented by scored vulnerabilities, which are the reference to evaluate the countermeasure later in this work.

Vulnerability/Attack Pattern The vulnerability and attack pattern representing the generalised concept of the previously identified security issues (given in Section 4.3). According to Shaw et al. [153], this is represented as a descriptive model. The inputs for this artefact are the two vulnerabilities/attacks from the previous vulnerability/attack assessment. Furthermore, the concept proposal maps the given information to threat categories in order to define them on a general level. A consideration of other system utilisation paradigms will conclude this pattern generalization.

Penetration Test and Proof-of-Concept The penetration test and PoC instantiates a CImpl of the several proposed approaches and concepts based on commodity embedded hardware. According to Glass et al. [53], this is represented as a system concept or approach. The input for this artefact is given by the vulnerability analysis and the countermeasure approaches. The presupposed hypothesis and the effectiveness of the countermeasures are tested through a LE.

¹⁵Compare Section 2.1

¹⁶Compare Section 2.2.2

¹⁷Introduced in Section 2.2.1

Mitigation Strategy The mitigation strategy artefact endeavours the conceptual strategies in hardening AMP-systems. It, therefore, conceptually analyses the relationship of vulnerability/attack and mitigations in order to be able to suitable requirements. According to Shaw et al. [153], this is represented as a descriptive model. Here, the outputs are the requirements as part of the risk treatment.

Countermeasures The countermeasures represent the solution (TSF) of the previously mentioned artefacts and seek to address the identified security issues. According to Glass et al. [53], this is represented as a system concept or approach. The solution space for the countermeasures is limited to the utilization of the intermediate level memory map capability. The countermeasures are evaluated by the security evaluation process¹⁸. The output is an effectiveness factor.

2.4 Summary

This chapter profiled the fundamental concept of AMP-based systems. Furthermore, the assessment process to identify, measure and evaluate security risks have been discussed. The chapter concludes with a description of the research methodology. Primarily, the following research questions are formulated due to the previous problem analysis and the foundation review.

In contrast to traditional virtualization schemes, which involves a virtual machine monitor or a hypervisor, the AMP approach does not intent to implement that kind of software instance between the multiple OS and hardware. However, the intermediate layer comprises of the configuration the underlying hardware layer. If it is assumed that the hardware design cannot be changed, the possibilities to eliminate vulnerabilities are limited to the higher layers. The following research aim has therefore been formulated:

Research Aim: *Harden an AMP-based system concerning the specific circumstances on the intermediate layer.*

The aim should be reached by keeping the actual AMP characteristic. According to the research aim, the following research questions have been formulated.

Research Question 1: *What is the pattern of DMA and cache-thrashing attacks reflecting the particular AMP characteristics?*

¹⁸Introduced in Section 2.2.2

Research Question 2: *How can one mitigate the risk of exploitation of the previously identified vulnerabilities?*

Furthermore, the research artefacts and methods are detailed. Primarily, developmental methods are applied. These include formulative and descriptive methods. Accordingly, the research artefacts are descriptive and formulative models. Case studies are conducted to show the applicability concerning two cases. The evaluation of the resulting proof-of-concept implementation is conducted by observing laboratory experiments.

Risk Assessment: Driver Information System

A Case Study of a Mixed-Criticality System for Vehicles

Contents

3.1 ToE- and Context Profiling	62
3.1.1 Mixed-Criticality Systems: A Case for AMP	62
3.1.2 System Decomposition	64
3.1.3 Experimental Platform	69
3.1.4 Threat Context Analysis	74
3.2 Threat and Attack Analysis	83
3.2.1 Threat Analysis	83
3.2.2 Attack Analysis	86
3.3 Risk Analysis	90
3.3.1 Impact Analysis	90
3.3.2 Attack Potential Analysis	92
3.3.3 Risk Definition	94
3.3.4 Hypothesised Attacks	94
3.4 Summary	95

Example is not the main thing in influencing others. It is the only.

Albert Schweitzer

From a functional perspective, AMP is just one particular way to empower a system with hardware utilization and separation capabilities. It does not matter what functions or features are executed by the AMP-based platform. However, an applied example warrants and legitimizes the concept. Therefore, this chapter elaborates on a case study through realizing an AMP MCS in the automotive context. Given that example, the risk assessment (compare with Section 2.2.2) is applied in order to derive the risks that are investigated further throughout this work.

3.1 ToE- and Context Profiling

3.1.1 Mixed-Criticality Systems: A Case for AMP

Introduction of Mixed-Criticality Systems

MCS is a particular type of system which aims to combine functions that are differentiated in their criticality. Various subcomponents which represent those functions are compiled together on a common hardware platform and aggregated in a greater context. The general advantage of such a system can be stated due in following:

"Most of the complex embedded systems found in, for example, the automotive and avionics industries are evolving into mixed criticality systems in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation and power consumption [...] how, in a disciplined way, to reconcile the conflicting requirements of partitioning for (safety) assurance and sharing for efficient resource usage." [21, p. 3]

The authors state the problem of combining functions on one hardware platform while handling the assurance verification differently for each of them. This leads to the following definition of criticality by Burns et al. [21]:

"Criticality is a designation of the level of assurance against failure needed for a system component" [21, p. 3].

Accordingly, the definition of MCS is:

"A mixed criticality system (MCS) is one that has two or more distinct levels (for example safety critical, mission critical and low-critical)" [21, p. 3].

Here, the authors aim explicitly for safety-relevant assurance. Well-known examples are the Automotive Safety Integrity Levels (ASIL) defined by the ISO26262 [83].

MCS in Automotive Environments

The aggregation of features is an ongoing trend in the automotive industry. As it was mentioned in the previous section, the advantage of MCS has multi-dimensional motivations (cost, space, etc.). The result of such aggregation efforts has evolved into highly integrated control units. Most commonly they are referred to as DCU [162]. Particularly, examples such as the *zFAS*¹ show that this has become a reality in the automotive sector [76].

A further automotive application of mixed-criticality is ECUs combining In-Vehicle Infotainment (IVI) and Driver Information (DI) functionality. The first example has decent requirements towards processing power and multimedia features. This includes graphics power, connection versatility and audio capabilities. However, the second example builds primarily on safety or dependability in general. The cluster instrument which shows the speedometer, mileage, rotations per minute and telltales is an example for DI functionality. That information is considered to be safety critical and mandatory to the driver. Particularly the telltales, which are warning signs shown by the cluster instrument, imply a certain ASIL criticality.

The previous example shows that MCS not only handle functions which imply requirements such as timing or stability (compare with Goswami et al. [57]), but also all possible system quality aspects can be targeted in an automotive system.

Hence, within this work, the definition of MCS has been adopted to the broader functional field of the automotive domain. Accordingly, MCS is defined concerning system quality goals.

Definition 3.1 *MCS integrate multiple quality domains (MC-domains), each of which is designated to the assurance of system quality goals.*

On a logical basis, the question arises as to how to suitably organise the mixed-critical functions. Special separation provides the ability to compose functions with similar criticality requirements into the same MC-domain. However, in practice the

¹zentrale Fahrerassistenzsteuergerät - centralized driver assistance control unit

bundles of functions can be driven by several reasons. For example, integration issues during the development of the system might lead to a different distribution criteria [101].

MC-domains compose functions with a certain tendency to fulfil a particular system quality. This means that, one MC-domain might be composed of functions with a focus on *safety* and *reliability* and another domain could consist of functions aiming for *usability* and *performance*. In Figure 2.11, this assumption is exemplarily shown. The severity of this tendency of a certain function is commonly referred to as *criticality*.

3.1.2 System Decomposition

In general, the case study realizes a dual domain MCS. The driver information system has been utilised to serve as a concept implementation to provide context to the assessment conducted in this work. Furthermore, it is applied in the experimental platform in order to mount laboratory experiments as well as penetration testing. Furthermore, this system concept has been adopted in several publications outside of the research scope of this work [72–74, 100, 101].

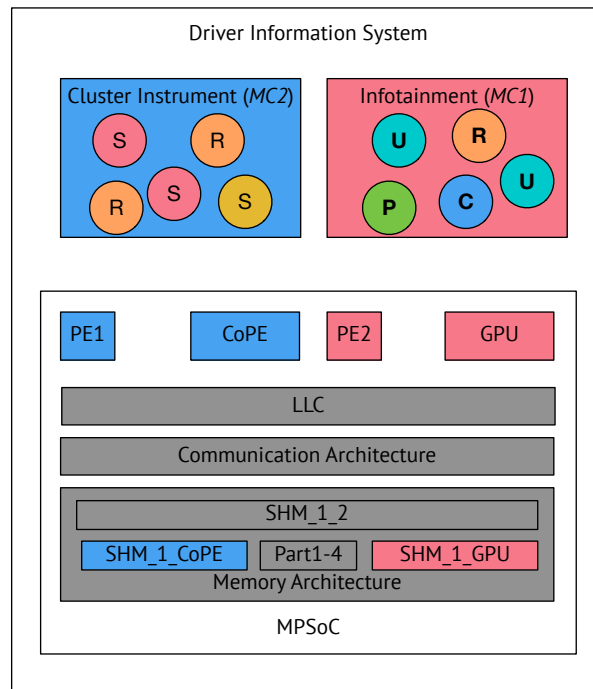


Fig. 3.1 Quality assignment for the Driver Information System.

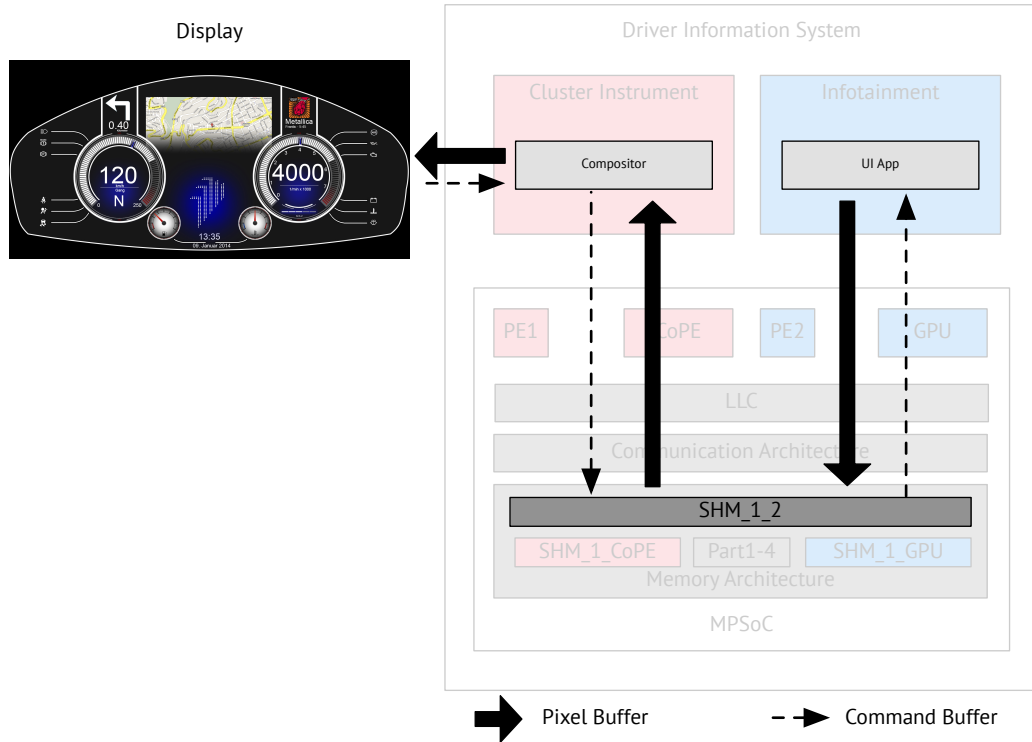


Fig. 3.2 Case Study: Driver Information System use case.

In general, the system composition consists of the two MC-domains. Referring to the definition of the ToE, an MC-domain realizes the concept of AD. These domains are denoted by *MC1* and *MC2*, respectively. Each MC-domain implies a main purpose. Here, *MC1* implements a controller for cluster instruments, whereas *MC2* facilitates a typical IVI system. The functional goal is to provide close-coupled graphical output. This means the output of each MC-domain will be composited and visualised on a common display. This display is located at the traditional position for cluster instruments, which is behind the steering wheel. Figure 3.2 visualises the interplay between the two MC-domains and the display. The communication is handled via a *shared memory* region in the main memory. There, a *pixel buffer* and a *command buffer* are located. Within the *pixel buffer* the graphical output is carried. The *command buffer* is used to transfer user interactions from the user interface back to the respective domain.

Function Architecture Each domain contains functionality which is typical for the applied area. *MC1* is considered to visualize the cluster instrument content and

information to the driver. These include: speed, mileage, gage, tell-tales², blinking lights and others. Furthermore, to display such information the domain includes functions to receive the information from an in-vehicle bus (such as the Controller Area Network (CAN)). According to the criticality consideration in *MC2* the vast majority of duties emphasise a criticality in safety and dependability.

In contrast, *MC1* includes functions with performance and usability emphasis. This is motivated by the fact that the IVI system implements functions such as media processing, radio, telephony and value-added cloud services. Notably, the latter involves connectivity interfaces such as Bluetooth, Wireless Local Area Networking (WiFi) and broadband Internet connections. In Figure 3.1 the criticality aspect is visualized.

Technical Architecture Technically, the setup incorporates system elements which are typical for MPSoCs in this area. The two MC-domains are assigned to the technical architecture, as shown in Figure 3.3. Three DMA capable PEs are considered. According to the ToE profile, these elements include a CA interface. A dual-core PE is incorporated, where each core is denoted by *PE1* and *PE2*, respectively. This dual-core PE is considered to be the application processing unit, executing the major software stacks of the MC-system. Internally, each PE incorporates a private level one (L1) cache and additionally, a second level cache (L2) is shared by *PE1* and *PE2*. Address translations are supported by the two-stage MMU concept, whereby the first stage is private to the each of the PE and the second stage is globally maintained. This aspect will be discussed later in this section.

The system includes a co-processor for special computation purposes. Within the automotive domain, this is often used to encounter real-time requirements or to make special hardware features available. Those include fault-tolerance technologies such as *lock-step* hardware failure detection and prevention [12].

Furthermore, a GPU is integrated into the system. In the functional scenario, the GPU handles all graphics rendering tasks. Main memory is represented by the *Memory* subsystem. For the sake of simplicity, *Memory* is considered to provide concurrent access. Arbitration of competing accesses is out of scope. The PE subsystems are connected through a simple interface to the CA. The CA itself is considered to be transparent to the PEs. It simply handles the issued data transfers. As it is for *Memory* the particular class or implementation of CA (NoC, cross-bar etc.) is opaque for this consideration. As a result, the PE address memory through their CA interfaces.

²Tell-tales are small warning pictographs which are intended to warn or inform the driver about the state of the vehicle.

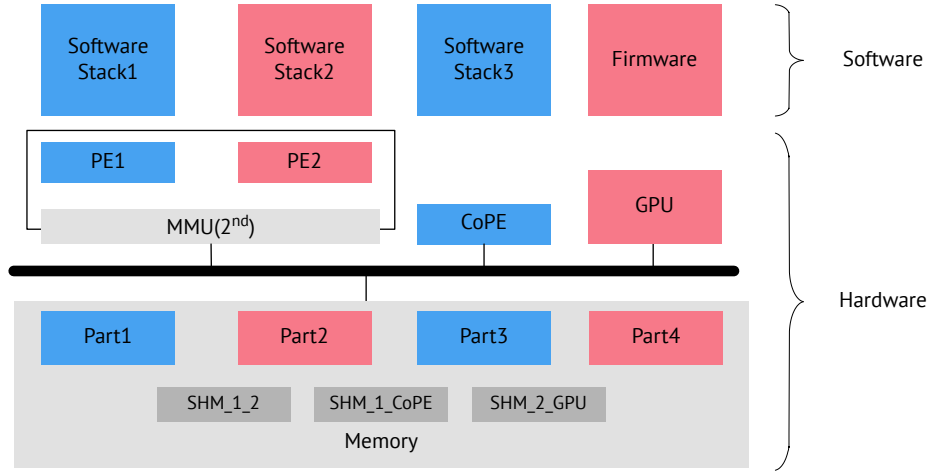


Fig. 3.3 Technical architecture and software stack assignments.

Further aspects shown in Figure 3.3, are the logical software-stack assignments of the particular hardware elements. Each PE is assigned with its software-stack. Internally the software-stacks consist of different system levels, as described in Section 2.1.2. The software-stacks of *PE1* and *PE2* are considered to contain the main functions of the system. Therefore, they consist of a rich OS. The *Companion Processing Element (CoPE)* is run by a firmware OS and a thin application software stack. The GPU is provided with firmware as well. Within the *Memory*, each software stack is assigned to a private memory partition. The partitions are denoted by *Part_i*. Furthermore, a Shared Memory (SHM) partition is reserved for communication purposes between the software-stacks. The logical MC-domain separation is shown in detail in the system memory map. Figure 3.4 depicts the two MC-domains and the system address space. In this simplified address space overview, the memory partitions are shown as single rows. Every row represents a particular block of memory. In this simplified view all memory blocks are equal in size. The coloured row in the corresponding column indicates whether the MC-domain has access to the memory range.

General Data Flow Diagram. Figure 3.5 illustrates the data flows between the technical system elements.

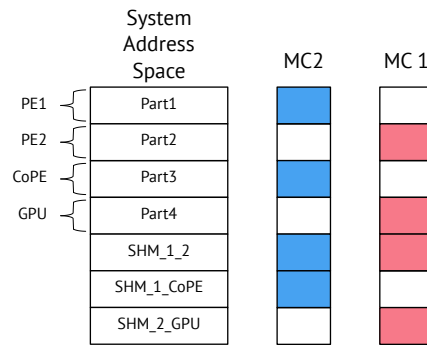


Fig. 3.4 Memory map overview.

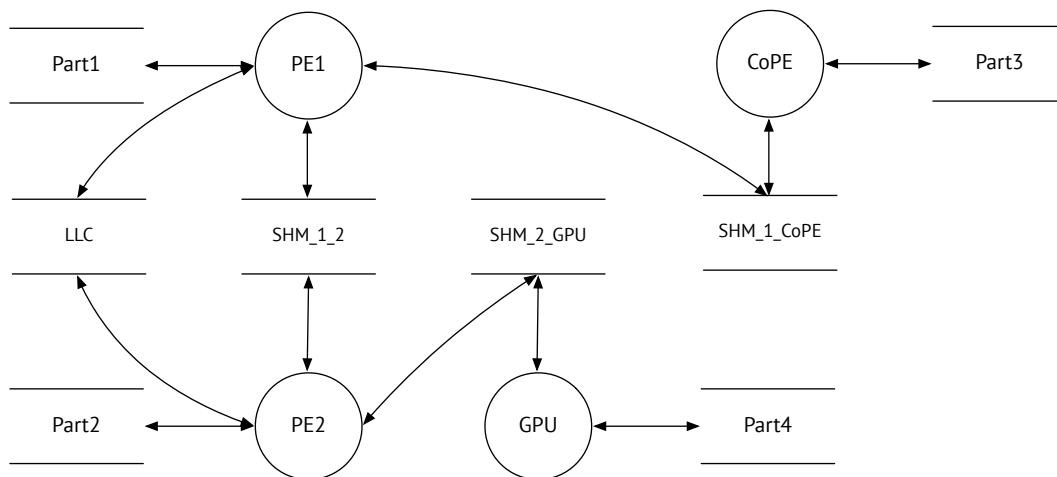


Fig. 3.5 Data flow diagram of the driver information system.

Operational Environment

From a broader perspective, a driver information system is part of a system of systems. The previously described system will be integrated into a vehicular environment. Accordingly, this system of systems implies a certain complexity which can be differentiated in several aspects. There are topological views on the system which represent the technical segregation of the systems, sometimes referred to as E/E architectures. They differ over the range of OEMs and their products. Usually, it consists of several ECUs connected through diverse automotive bus systems. From a functional viewpoint, there are distinct domains which categorise the function of a system. Functional domains make segments for the intended purpose of the ECUs. As an example, there is a domain for the powertrain ECUs. Those include ECUs such as engine controllers,

wheel speed sensors, and other actuators. To extend this example, most commonly, there are usually domains for infotainment, body control and ADAS as well. The infotainment domain includes functions such as navigation and telematics, just to name a few. Body control domains might include a Body Control Unit (BCM) ECU handling the wireless key access to the vehicle, for example. The ADAS domain might implement automated driving functionality and incorporates the necessary sensors and actuators for that purpose. Figure 3.6 has exemplary E/E architecture and functional domains. Domains and topologies are independently segmented. However, often this is a common practice in E/E topologies.

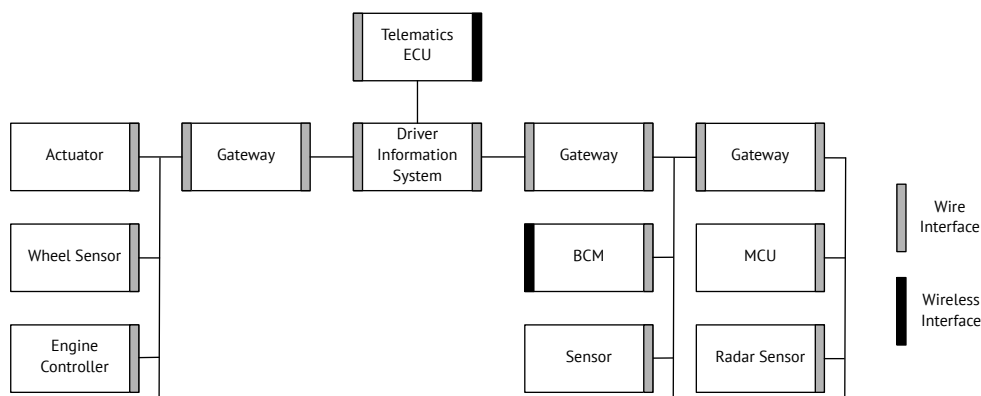


Fig. 3.6 Exemplary E/E architecture and functional domains.

3.1.3 Experimental Platform

Mainly, the aim for selecting a suitable hardware platform (MPSoC) is to enable *reproducibility* and *transferability* of the given results and concepts. Hence, a commodity platform was chosen which is available to the public domain. A MPSoC platforms is typically an integrated set of IP-cores (hardware elements). The integration, in other words, the actual platform, is proprietary and therefore a specific facility. Nonetheless, the integrated IP-cores implement common or widespread architectures. What this means is, as long as the compilation of the hardware elements follows the ToE definition, the aspects and results of this work are reproducible and transferable.

As a result, to select the appropriate platform these two aspects have to be considered. The particular platform facility must be available including all necessary documentation and sources to implement and operate the experiments. This enables their reproducibility. Transferability relies, in this case, on the utilisation and architecture of the IP-cores. They must apply universal architectural concepts which in

turn are widespread. If this is not be given, the results are only applicable to a limited amount of systems. In the following sections, the experimental platform is described in detail.

Texas Instruments OMAP5432 EVM

Platform Introduction

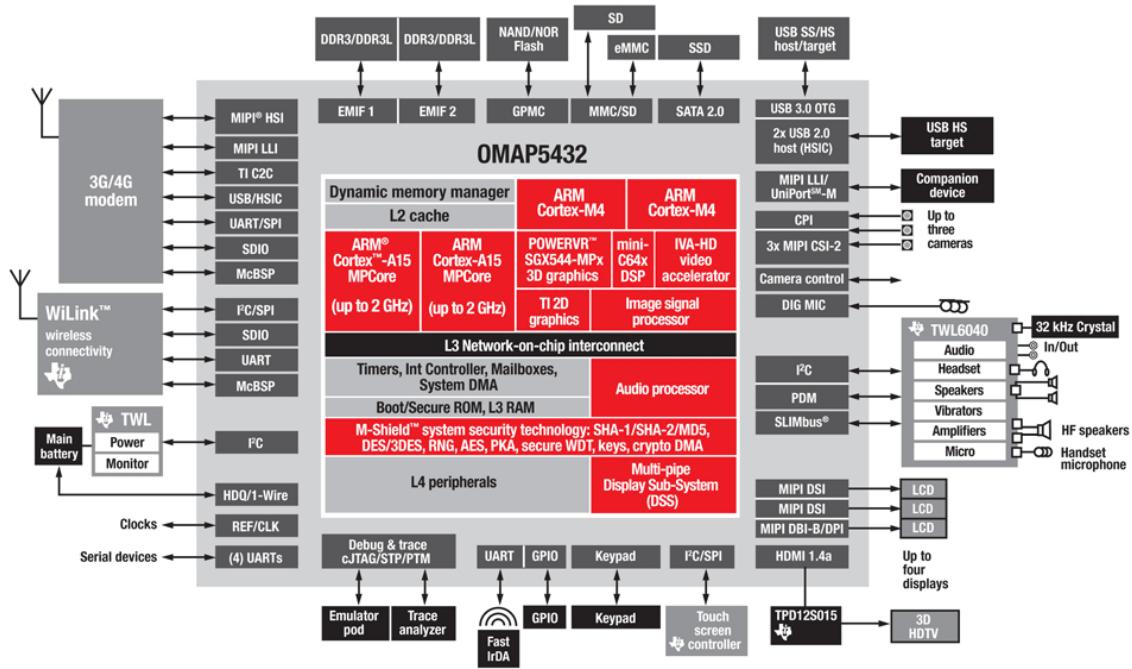


Fig. 3.7 Texas Instruments OMAP5432 EVM [79].

In this work, the Texas Instruments OMAP5432 Evaluation Module (EVM) [79] has been chosen. The OMAP5x platform was one of the first to incorporate multi-core PEs with virtualization extensions. This feature and the broad range of further IP-cores and peripherals made it into a suitable hardware platform, particularly for the driver information case study.

"OMAP5432 EVM is an OMAP5432 ES2.0 platform designed to provide access to as many of the powerful features of the OMAP5432 Multimedia Processor as possible, while maintaining a low cost. This will allow the user to develop software to utilize the features of the powerful OMAP5432 processor." [79, p. 9]

In addition to the range of functionalities, the manufacturer provides a reasonable software development kit in order to use the system out-of-the-box [171]. Here, the Linux and Android Board Support Package (BSP) are worth mentioning, since they are utilized in the PoC implementation. However, the availability of QNX and Greenhills INTEGRITY BSPs is valuable for researchers seeking domain specific operating system evaluations.

The EVM is equipped with the OMAP5432 MPSoC of Texas Instruments. Amongst other external peripherals, the EVM features 4GB of Embedded Multi Media Card (eMMC) non-volatile flash memory, 2GB DDR3L³ volatile random access memory. The entire specification is listed in Appendix A. Furthermore, Figure 3.7 visualizes the features of the EVM.

OMAP5 General Features

The OMAP5432 MPSoC embodies the heart of the EVM. It integrates a plethora of IP-cores, of which the ARM Cortex-A15 microprocessor units (MPU)⁴ is the central PE on this platform. Further PEs are incorporated. Here, the Image Processing Unit (IPU) and the 3D-graphics accelerator are prominent examples, because they are utilised to demonstrate some of the attack approaches. A detailed visualisation of the entire MPSoC architecture is given in Figure A.1. In the following, a brief overview of the MPSoC is given [169, p. 288]:

- CortexTM-A15 microprocessor unit (MPU) subsystem, including two ARM[®] Cortex-A15 cores
- Digital signal processor (DSP) subsystem accelerator, hardware virtualization support, and large physical address extensions (LPAE)
- CortexTM-M4 image processing unit (IPU) subsystem, including two ARM Cortex-M4 microprocessors
- Audio back-end (ABE) subsystem
- Imaging subsystem (ISS), consisting of image signal processor (ISP) and still image coprocessor (SIMCOP) block
- 3D-graphics accelerator subsystem, including POWERVRTM SGX544 dual-core

³Dynamic Data Rate type 3 - low voltage SDRAM

⁴Within the Texas Instruments documentation, the microprocessor unit is abbreviated with *MPU*. However, this collides with the abbreviation of the memory protection unit within this work. Throughout this text, it will be made clear by the context which definition applies.

Cortex-A15 MPU

Particularly, the ARM Cortex-A15 design [9] made it possible to introduce virtualized systems in embedded environments. The design facilitates the ARM Virtualization Extension (VE) [119]. The ARM VE feature a certain set of functions which are previously mentioned in Section 2.1.4. These features include [119]:

Hypervisor execution mode: Additional privilege level which is higher than supervisor mode. This will enable the VMM to execute at a higher privilege than the guest OSs, and the guest OSs to execute with traditional operating system privileges, removing the need to employ Paravirtualization techniques.

Interrupt provisioning: The provision of mechanisms to aid interrupt handling, with native distinction of interrupt destined to secure monitor, hypervisors, currently active guest OSs or non-currently-active guest OSs. This will dramatically reduce the complexity of handling interrupts using software emulation techniques and shadow structures inside the VMM.

Two Staged System MMU: The provision of a System MMU to aid memory management, that supports: multiple translation contexts for multiple DMA capable masters, two levels of address translation and hardware acceleration and abstraction.

Despite the VE, the Cortex-A15 implements the ARM architecture version 7 [10]. Due to the advent of 64bit architectures in embedded devices, the successive architecture ARMv8, the concept of the VE remains similar to the 32bit version. Particularly, concerning the transferability aspect, the ARMv7 is the most implemented processor architecture on the MPSoC marked for mobile devices. Superficially, the aspect of demonstrating attacks on common or commodity hardware comes into focus when it comes to transferability. The broader the spread of the architecture, the broader the impact of the investigated attack. However, the given concepts are based on general design principles rather than industrial implementations.

Software Stack

In order to provide some level of reproducibility, all results have been created using open-source software. In the following, the major components are briefly described.

Operating System The software stack is comprised of a fairly common Linux distribution. The implemented Kernel Version is Kernel 3.8.13. EVM specific patches were used from the OMAP repository [177]. Only the kernel-space of Linux is used to perform the software-stack level approaches. Accordingly, the description of the userland implementation can be omitted, because it is not needed to reproduce the shown concepts.

Bootloader The system bring-up is implemented by utilising the well-known *u-boot* bootloader [36]. U-boot aims at being simple, fast, portable and configurable. Due to the wide acceptance and widespread hardware support, this bootloader is suitable to facilitate the PoC. The EVM specific patches and configurations have been taken from the EVM development kit [177].

Boot Sequence

In the following, the initial boot sequence of the experimental setup is described. The boot sequence is a crucial part of the setup of an AMP-system because in this phase, the system will be configured. In Figure 3.8 the boot sequence is visualized in a sequence diagram. *PE1* is considered to act as the boot processor in this example. Accordingly, this processor core loads and executes the Initial Program Loader (IPL). An IPL is usually a particularly small portion of code that performs the fewest steps necessary to configure the bare hardware [138]. IPLs, hence, are delivered along with the hardware.

The sequence starts off with the reset of the hardware. *PE1* invokes the IPL which loads and invokes the *u-boot* bootloader. All necessary configurations of the hardware are done by the *u-boot* program. One of these initializing steps is the setup up the MMUs which enforce the memory isolation of MC-domains when the system is running. During this step, the memory mappings are created and stored in memory. The last step is to load the OS images, which are, in this case, Linux kernels. Furthermore, configuration files (device-tree-blobs) to set up the kernels are loaded. Right after the loading is finished, the kernels can be invoked. During the initial boot sequence, *PE2* is idling in a wait for reset state. With the Kernel invoked, the processor cores start the boot sequence of the Linux kernels.

Due to the sake of simplicity, the boot sequence shown here lacks the verification and attestation of the booted images. For a proper secure boot sequence, the control flow must not be handed over to the code without a successful verification of the

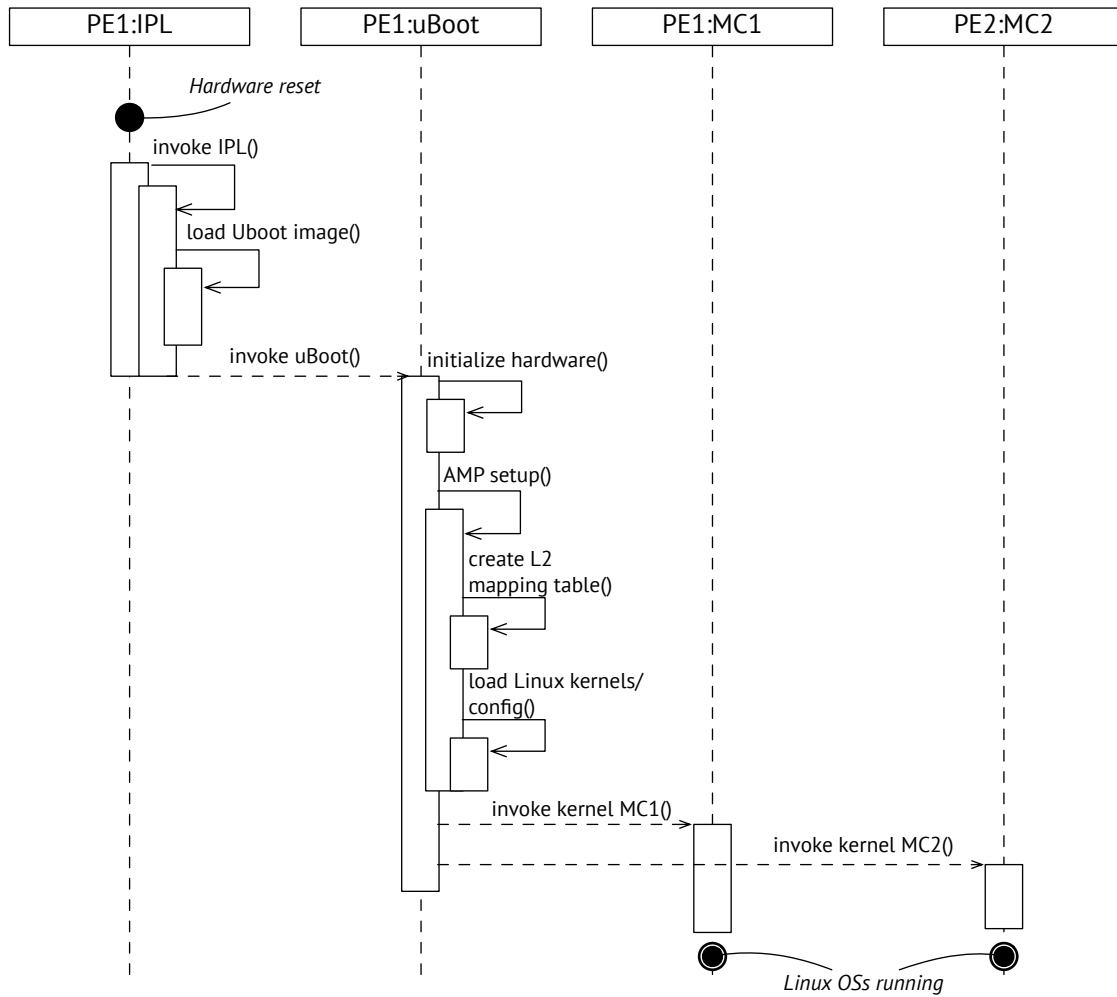


Fig. 3.8 Boot sequence of the experimental system setup.

authenticity of all binaries that are to be executed. In state-of-the-art automotive ECUs this would be mandatory in terms of security.

3.1.4 Threat Context Analysis

Security considerations always deal with the question of what are the security relevant elements of the system. The answer to this question is rather simple: everything is security relevant. Security stands orthogonal to anything in a system. In other words, security is a non-functional requirement that can be expressed as a quality constraint. A method to determine such constraints is threat analysis. Since every functional or technical element of a system is affected by threats, this is a complex and demanding effort. Particularly, the challenge is to identify the specific elements of concern with

regards to the appropriate abstraction of the system and the state of the system in which it is threatened.

Context and Assets

Although this section encompasses offensive aspects of the ToE, in the beginning, a fundamental aspect needs to be discussed. This is the question: what are the subjects of this analysis? In other words, what are the assets to be assigned with a protection goal? For sure, an asset is an entity with a certain value to one of the stakeholders of the ToE. As it is described in Section 2.2.1, the particular STRIDE threats will be applied by the *STRIDE-by-Element* pattern.

Stakeholders. In this work, two stakeholders are considered: first the OEM, which makes the vehicular systems and second, the driver who uses the vehicle as the end user. The value chain or supply chain in the automotive environment is large. Several levels of suppliers (tiers) are involved. For the sake of simplicity, this context will be represented by the OEM stakeholder. Furthermore, the end user (driver) will represent the human beings in the driving environment.

Assets in MC-systems. As it was stated before, an impact analysis shall reveal or help to identify what the important assets of a potential target are. On a high level, those assets are categorized by *safety*, *financial*, *operational* and *privacy* (compare impact severity definition in 2.2.1). Applied to automotive scenarios, a high-level asset on safety is the physical integrity of the driver and other traffic participants. The physical integrity of a human-being is one of the most significant concerns when it comes to vehicles and their safety relationship. Also, from the OEMs perspective, the financial aspect of their product turns into focus as well. Business cases must be secured. A good example of "lost" business cases is if a charged feature is fraudulently activated by an end user. In an environment which is subject to an evolving modularization of functionality, this becomes a serious case. As a consequence, the financial aspect of protecting business cases is a high-level asset for the OEM. Moreover, there are much more examples applicable to this environment.

MC-domains are logical assets. The system model in Section 3.3 consists of a functional and logical view on the system. On a higher level, there is an MC-system which consists of several MC-domains. A specific MC-domain represents the level of granularity this threat analysis refers to. MC-domains are introduced to compose

functions with a certain criticality. Therefore, since it has a defined criticality, it implies a certain impact to the stakeholders if the quality of this particular MC-domain is not met. If the MC-domain is compromised, the impact is high.

Hardware elements are technical assets. On the hardware, technical level (compare Section 3.1.2) the assets are the single hardware elements on the MPSoC. This is true for all elements that relate to the intermediate level. In fact, all those elements are addressable by the system memory map: PE, memory, peripherals etc. This selection is motivated by the fact that these elements on the level given in the system model technically facilitate the actual Mixed-Criticality (MC)-domain. If the ultimate goal is to analyse a particular MC-domain, the technical facilities need to be assessed.

Asset Summary. Ultimately, a single MC-domain is concerned in this analysis. The analysis is centralised to the identified assets. Therefore, the threat analysis seeks to determine the threats on this level first. In the following, the intermediate level system elements are assessed in order to lead into the attack analysis.

Entry Points

External Threat Surface. The vehicular attack surface is wide-spread and complex. Despite the past decades of automotive engineering, attacks have turned into the focus of manufacturers and suppliers because interference is a crucial factor for the dependable and reliable operation of their products. This is motivated by the fact, that the number of wireless interfaces into the car evolved substantially.

External connections to the environment from a functional point of view include two main areas: services and functions, which add value to the inner-vehicular system and Consumer Electronics (CE) oriented infotainment services. The latter include general purpose Internet access, which opens the entire world of connectivity services. Those services include social media such as communities, messaging and data sharing. Furthermore, media access to audio and video streaming are an important functionality to be integrated into the vehicular system. The area of value-added services which are provided by the manufacturers or operators includes: remote software updates or Over-the-Air (OTA), map and positioning services such as high definition maps for automated driving functions, fleet management services for the commercial car sector, remote diagnostics, Car2X⁵ communication and Emergency Call Interfaces (eCall) [125]. Notably, the value-added services have potentially a high impact on the function

⁵Abbreviation for Car to something else.

of the vehicle. For example, remote software updates are naturally prone to bring illicit functionality directly into the vehicle.

Technically, all of the services as mentioned earlier need to apply a technical interface to communicate to the outside world. Common technologies which are used in this area are WiFi, Bluetooth and cellular networks such as LTE. However, the technical interfaces are not limited to such wide and near range general purpose communication technologies. In the automotive environment, further wireless channels are implemented. Those include keyless and wireless entry systems and wireless sensors such as wheel speed odometry.

In addition to the wireless interfaces, cable connections are still a crucial aspect. Aside from media devices which are plugged into the IVI system, for example, diagnostics accesses are also important to mention. The latter are usually implemented using the standardised On Board Diagnostics (ODB)-interface or the Joint Test Action Group (JTAG) interface of an ECU. The former includes USB pluggable media, smartphones, or optical media such as CD, DVD or Blue-rays.

Attack Goals. Before the particular relationship to attacks is elaborated upon, this section aims at building the overall context for attacks on vehicular systems. In order to mount attacks on the technical MC-system, the threat agent needs to break several horizontal or vertical security controls to reach out to the ToE. Although these steps are out of scope of this work, they are discussed very briefly to set the overall context and makes it easier to delimit the contribution of this work.

In Section 3.1.4, the vehicular environment is described. There are all entry vectors (interfaces) into the vehicle shown. In Figure 3.9, the presumed attack-tree to break into the MC-system is shown.

The attack goals are derived from the three major threats of the previous chapter, which are *Denial-of-Service*, *Tampering* and *Elevation of Privilege*. These threats are taken over and shown as generic attack methods (dashed orange rectangles).

For the sake of simplicity, the attack tree is condensed and encompasses two root nodes, each of which partially shares the same nodes on the lower levels.

The specific attack goals are illustrated in the red rectangles. Either adversary seeks to degrade or disrupt the performance or takes control of a particular function in a target MC-domain. The latter refers to tampering and EoP threats and the former to DoS.

The following sub-notes towards the root, comprise of the actual scope of this attack assessment. In general, the intermediate layer must be exploited. These designated

methods are dependent on the two roots (goals) of the attacker. Here, these are denoted by *Stall-PE* for the DoS threat and *Breach Memory Protection*.

Furthermore, the attack tree shows two initial events (blue circles) which are assumed to be done in advance. It is obvious but yet not trivial that the attacker is required to *gain entry* into the MC-system first. In parallel to gaining entry, the adversary must at least control a software-stack of one of the MC-domains. In order to reach this goal, it might be necessary to exploit the application layer as well. Most commonly, such kind of elevation of privilege exploits are initiated from applications within the highest layer. As an illustration, at this level, the data is handled which was sent over a communication channel and therefore a severe attack surface. Nevertheless, it might be possible that vulnerabilities within the software-stack are remotely exploitable such as, improper implementation of the communication stacks, such as shared memory, Internet Protocol (IP) or even CAN.

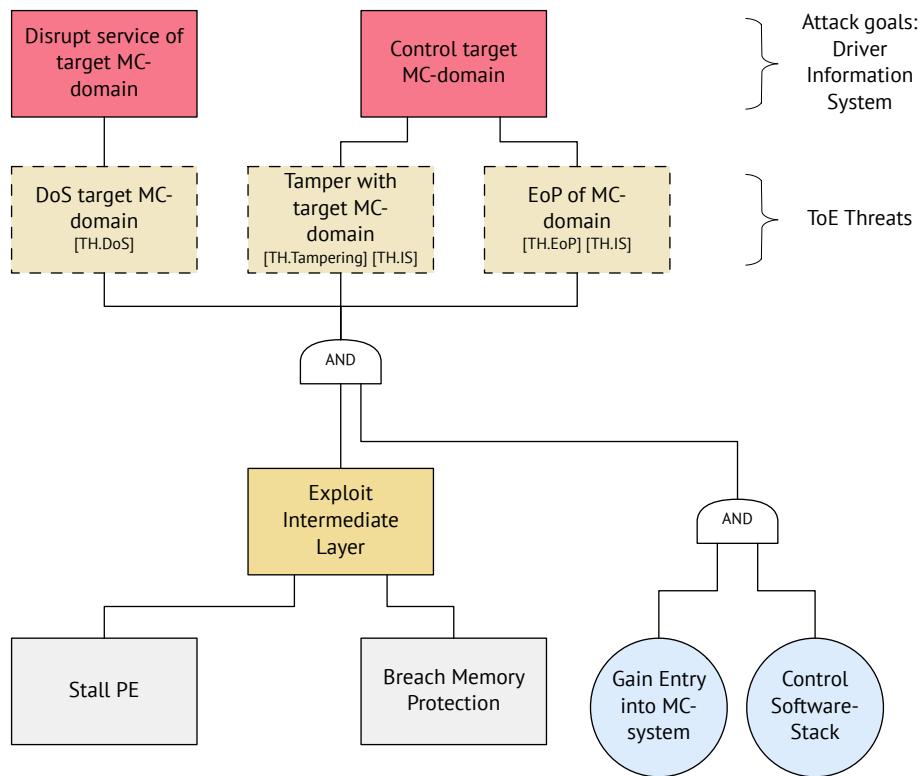


Fig. 3.9 Presumed path into the MC-system. ToE threats defined in 2.1.5.

Threat Agents

To estimate the likelihood of a threat being exploited, adversaries have to be taken into account. Generally speaking, a threat agent is an entity acting adversely on the ToE. The root of such adverse actions are usually human beings. They have an intent and follow a motivation to gain benefit out of an adverse action. In compliance with [49, p. 51] the following threat agent categories are considered:

- **State Agents:** State-sponsored national groups that are engaged in espionage and either clandestine or covert action. Examples are national intelligence agencies.
- **Cybercriminals:** Criminal enterprises or loosely organized group of criminals that are usually well-funded. Common examples are black-hat⁶ hackers.
- **Ethical Hacker:** Enterprise or loosely organized group which is well funded and skilled. Common examples are researchers and white-hat⁷ hackers.
- **Non-Privileged Insiders:** People that have to overcome resistive controls in order to affect harm. Examples are end users such as the driver or owner of a vehicle.
- **Privileged Insiders:** Group that has legitimate access to resources of the ToE. Examples are employees and car garages.
- **Malware:** Non-human means which acts autonomously to a certain extend. Most commonly this is initiated by cybercriminals.

The threat agent categories mentioned earlier are distinguishable by characteristics such as *motivation*, *primary intent*, *sponsorship*, *capability*, *preferred targets*, *personal risk tolerance* and *concern of collateral damage*. As an example, an end user might want to activate features in his infotainment unit to save money fraudulently. However, his capabilities are low. In contrast, an ethical hacker, such as a researcher, has much more resources and knowledge to compromise the feature activation, but his motivation is different in this regard. Researchers aim for reputational aspects of motivation.

In this work, we assume threat agents with rather high capabilities, including the resources and knowledge to compromise a system which is under consideration. This includes the knowledge of concepts in mixed critical systems, access to restricted

⁶Black-hat hackers aim at gaining benefit to the disadvantage of others.

⁷Activities of white-hat hackers aim for the education of the advantage of others.

documentation and access to the ToE as well. It is assumed to deal with a mixture of *privileged insiders* and *ethical hackers* with high motivation.

Presumed Threat Agent Capabilities

This section elaborates upon the attacker classification which has been assumed. Assumptions on the attackers are an essential key. On the one hand, it helps to set the level of detail of investigations and analysis, and on the other hand, it serves as a factor to choose suitable countermeasures for certain attacks. As is mentioned in the previous Section 3.1.4, attacker classes range from non-privileged insiders, to white-hat hackers to state agencies. They differ regarding their motivation, intention, resources, risk tolerance and expertise. Although in this consideration only the technical skills will be considered in detail, some examples of objectives of attackers are given.

Particularly, in the automotive domain, safety is a crucial impact aspect for security considerations. Intentionally, safety might be the aspect which comes to someone's mind first when it comes to protection from attacks. The range of possible incidents is wide, particularly regarding the physical integrity of traffic participants or the driver. As an example, within the impact analysis, the higher severity levels expect an uncertainty of survival of the vehicle's occupants. An attacker could compromise one of the actuators of the vehicle and take over the longitudinal or lateral driving function. By compromising the actuation, an attacker would have direct control over the car, which allows for the conclusion that the actuation has the most severe impact on the safety of the effected targets. However, by attacking the sensing and furthermore the computational part of the vehicle, an adversary might be able to affect the motion of the vehicle as well. It can be assumed that an MC-system will more likely be implemented to combine a large set of abstracted functionalities rather than to physically control actuators directly. The objective of an attacker might be to interfere with the physical integrity of human beings or physical things in order to blackmail others, for example. This is indeed a very dark scenario. Nevertheless, if an attacker could show that they are capable of doing so, they might reach their goal without actually mounting an attack in the real world. An evenly important category of attack objectives is the financial side. This can be considered for the driver and even more critical for the producer of the vehicular system. Attackers could aim for intercepting payments which are issued to assign a value-added-service. So, the objective is to gain a financial advantage. For the driver or owner of the vehicle, this has a direct impact on their financial situation. Also, for the OEMs or tiers, finance might be affected directly or indirectly. For sure, attackers could intercept payments handled

by their vehicular or backend infrastructure as well. More severe are the financial impacts caused by regulatory fines due to non-compliance with regulations, for example. Due to an illicit configuration, system limits could be manipulated, and the vehicle could lose its operating license. Attacks on the operation of the vehicle generally aim for the degradation of the performance of the system. Adversaries might aim for an annoyance of the vehicle's owner or to deny the service of an entire fleet of a particular manufacturer. A good example can be adopted from the private computer environment. So-called ransom ware aims at encrypting the user's data on its hard drive to make it only accessible by the originators of the malware. Only after a certain payment can the personal data be decrypted and made accessible again. Privacy aspects becoming more and more into the focus of users, on the one hand, and attackers, on the other. Particularly, value-added services which are potentially connected services might reveal plenty of data to create profiles of the vehicle's owner or driver. For example, the vehicle could be tracked, or private data such as identities can be collected and stolen. For adversaries, there is a broad black market to turn private data into revenue. Attackers might also aim for degrading the reputation of a particular manufacturer. Through the introduction of any of the mentioned attack objective categories, their reputation would be indirectly degraded.

Capability Factors

In the following, the technical aspects of the threat capabilities are elaborated upon. This aims at creating an attacker profile based on the concept of quantifying risk (compare 2.2.1).

Knowledge of the targeted system is a crucial aspect when one is considering the potential of an attacker. In general, knowledge means the awareness of system internals. This includes technical documentation of software and hardware as well as the functional behaviour. As an example, the attacker might be aware of the communication infrastructure of particular MC-domains. Moreover, it is considered to be known how to configure the hardware to create the MC-system. Indeed, some of the information about the SoC architecture is only available through non-disclosure agreements with the manufacturer, which is even more the case for security-related information. Nevertheless, to follow a security-by-obscurity principle has been proven to be a bad strategic choice [118]. As a result, it is considered better to defend against adversaries who are aware of the entire system construction.

The level of expertise describes the attacker's ability to conduct an adverse action against the target. In contrast to the knowledge of the system, which aims for the available information, the expertise helps to gain unknown information. Reverse engineering skills are a good example for gaining information about the target. Moreover, it is not only about the information of the system, but it is also the identification of vulnerabilities. Namely, the vulnerabilities will later be combined and formed into an attack vector. Not only the information about a target becomes certain, but also the actual act of compromising the system is an important factor to express the level of expertise of an adversary. This includes, for example, the ability to implement exploits or use the appropriate resources to do it. As a result, concerning the AP model, the assumed level of expertise is on the expert and multiple experts level.

Equipment is the actual mean of applying the expertise of the adversary. It is, therefore, a crucial factor to estimate capabilities. The factor is influenced by the degree of how tailored a means must be to be applicable for the purpose of the attack. For example, is it environment-specific or rather standardised? Cost is also an important factor. It makes a difference if the adversary needs to make huge investments. As a side note, this lowers the return on investment as well. The assumed resource level might also be seen as an expression on how much power an adversary can apply to an attack. The anticipated level of equipment is *bespoke*. The adversary is capable of crafting equipment for the specific ToE.

The window of opportunity is treated as unlimited. Usually, the window would limit the time range in which the target might be accessible to the adversary. In some scenarios, this is a crucial factor, mainly when feasibility needs to be considered. As an example, in a situation when the attacker seeks to gain remote access to a vehicular system, he has several possibilities, as described in 3.1.4. Considering a wireless interface, an adversary could spoof the roadside broadcast antenna, for example. It would be necessary to be in the physical range of the target vehicle. Hence, the window of opportunity is limited to the physical broadcasting range of a wireless transmitter to get a connection to the vehicle. By considering an MC-system, it is assumed that the attacker has already gained access to the system. Once the adversary has compromised an MC-domain, the window of opportunity is unlimited.

Summary of technical capabilities It is presumed that the adversary has the following capabilities:

- Compromise an MC-domain on all logical and technical layers. Practically speaking this means that the adversary fully controls the hardware and SoC (hardware elements) which have been assigned to the MC-domain by purpose. To put it in another view, the adversary inherits all technical capabilities the layers he has compromised have. This includes, for example, access privileges and address mappings. This assumption is reasonable due to the integration of connected services.
- Access to all necessary documentation of hardware and software running on the system. The attackers know exactly how the MPSoC is implemented. They are aware of the system memory map to utilise the hardware from the software layers. Furthermore, all software layers, reaching from applications, software frameworks, and operating systems are known to the adversary, either through re-engineering beforehand or due to its documentation.

3.2 Threat and Attack Analysis

This section provides an overview of threats applicable to the DFD shown in 3.11. The section is split into three subsections which elaborate on high-level threats based on a logical system architecture. This allows for the showing of the threat scope. Within the subsequent subsections, this scope is analysed in further detail. The threats are presented per interaction following the *STRIDE-per-Element* as discussed in Section 2.2.1.

3.2.1 Threat Analysis

In order to consider the threats to a single entity, such as a MC-domain, it makes sense to assume another entity of equal type. For sure, a MC-system consists at least of two or more MC-domains. Therefore, herein are two MC-domains considered. One for the adversary and one for the asset (victim).

Logical Data Flow Diagram. In Figure 3.10 the logical DFD is illustrated. The MC-domains are denoted by *MC1* and *MC2*. To illustrate the communication of the two domains, two data flows are incorporated. One of them (*HMI commands*) represents

the ability to trigger Human Machine Interface (HMI) events in the infotainment domain. The other data flow, denoted by *Rendered Pixels*, represents graphics that are transferred to the cluster instrument domain (MC1) which displays the video output. As a result, data flows bi-directionally. This means that each of them implies inbound and outbound data traffic.

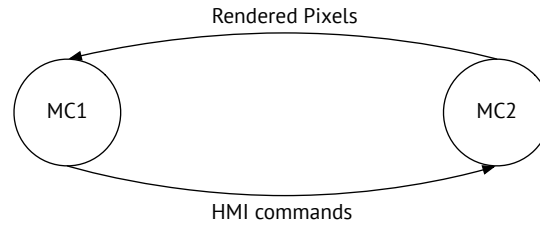


Fig. 3.10 Logical Level DFD.

Furthermore, Table 3.1 elaborates on the applied threats in more detail. In this regard, the explanation includes practical examples on how such threats could be facilitated. Besides from this, the violated security goals or missing protection mechanisms are stated. This will lead to the attack assessment in the following step of the offensive assessment of the ToE.

Threat Scoping

Overall, the STRIDE model provides six classes of threats to be applied to an asset. However, not every threat is contributing to the attack goals defined in Section 3.1.4. Therefore, in this work and for the purpose of considering these attack goals the STRIDE threats are divided into primary, secondary and out-of-scope. Justifications are given on the basis of the primary threats.

Primary Threats: Tampering and DoS. The main focus is put on threats that concern the integrity and availability of functions. As a result, tampering and denial-of-service are the primary threats in this area. It is considered, that tampering is the root-cause for other, secondary, threats in such system compilations. For example, due to modified and therefore illicit data the privilege of a (DFD) process might be elevated as a result. Therefore, tampering leads to Elevation of Privilege (EoP).

Secondary Threats: Spoofing, EoP and Information Disclosure. Although spoofing is a common and very severe threat in many areas, as it is in MC-systems, on

Table 3.1 Listing of logical level threats, including a description of the violated security goals.

Type	Threat	Description
S	MC1 impersonates MC2. MC2 is impersonated by MC1.	An adjacent entity (MC-domain) impersonates by compromising a communication channel. Communication entities are improperly authenticated.
T	MC1 tampers with MC2. MC2 tampers with MC1.	MC-domain is tampered with by another by modification of data or control flow. Memory storage does not implement proper access control mechanisms.
R	-	Repudiation is not applicable in this technical scenario
I	MC2 discloses information of MC1. MC1 discloses information of MC2.	A MC-domain information has been disclosed by another. Information includes intellectual property or implementation details for reverse engineering. This is often induced due to improper access control.
D	MC1 disrupts the service of MC2. MC2 disrupts the service of MC1.	Mount a DoS attack by over committing shared resources. Resource accesses are not scheduled by a privileged instance.
E	MC2 elevates privilege of MC1 by compromising it's control flow. MC1 elevates privilege of MC2.	MC-domain is compromised with by another due to the exploitation of a vulnerability to elevate/escalate privileges. Improper isolation mechanisms (vertically/horizontally).

the intermediate level the in-personification of entire MC-domains is not concerned. This is only reasonable if the particular functions within the MC-domains would be considered. As a result, this is omitted in this consideration. In the case of EoP, it is assumed that this threat can be represented by tampering. Information Disclosure has many aspects in the automotive domain. Those reach from privacy to intellectual property issues. In this consideration, information disclosure is more or less treated as the ability to read data for the purpose of re-engineering of system internals. Therefore, it contributes to the reconnaissance phase of attack vectors.

Out-of-Scope Threat: Repudiation. Repudiation is a threat which is usually applied in complex scenarios involving security concepts such as trust in third parties.

In this mostly technical driven consideration, the security goal non-repudiation is not reasonable.

Focus Threats

MC-domain is tampered with by adjacent MC-domain. Possibilities of tampering with an MC-domain are broad. Many assets inside the MC-domain exist which are modifiable for any reason. Despite the previously discussed qualities, the domain has to satisfy the facilities of the MC-domain to imply reasonable motivations for threat agents. The root cause of being able to tamper with such facilities are, for example, missing access control capabilities. A crucial precondition to implement an access control mechanism is that the subjects and objects that are to be controlled are tangible. This means that the adjacent elements must be identifiable and therefore also differentiate-able.

MC-domain is disrupted by adjacent MC-domain. The disruption of an MC-domain concerning the MC-system model is considered to be inducted by resource sharing issues. For sure, in a highly integrated platform, a large surface exists where resources need to be shared among all domains. Those *touch-points* are not practically avoidable. There must always be a mechanism which is handling competing accesses. Nonetheless, if accesses are not administered, by implementing collision avoidance, for example, disruption effects could be enforced by the purpose of a threat agent. Furthermore, examples of those touch-points are not obvious in every case like shared memory is. Shared power supplies and caches are worthwhile to mention here as well.

3.2.2 Attack Analysis

The two threats that have been introduced previously, are now analysed in their technical context.

Technical Data Flow Diagram. The technical representation of the DFD includes the physical elements of the system model. In Figure 3.11, the DFD representing the driver information system is shown. Due to the complexity of this particular DFD, the detailed analysis is limited to the items which are drawn in solid lines. The items in-scope cover all possible threats concerning the system model. This means all technical elements such as PE (and multi-core PE), CA, memory and data flows are represented. All other elements are treated as adjacent entities which embody the

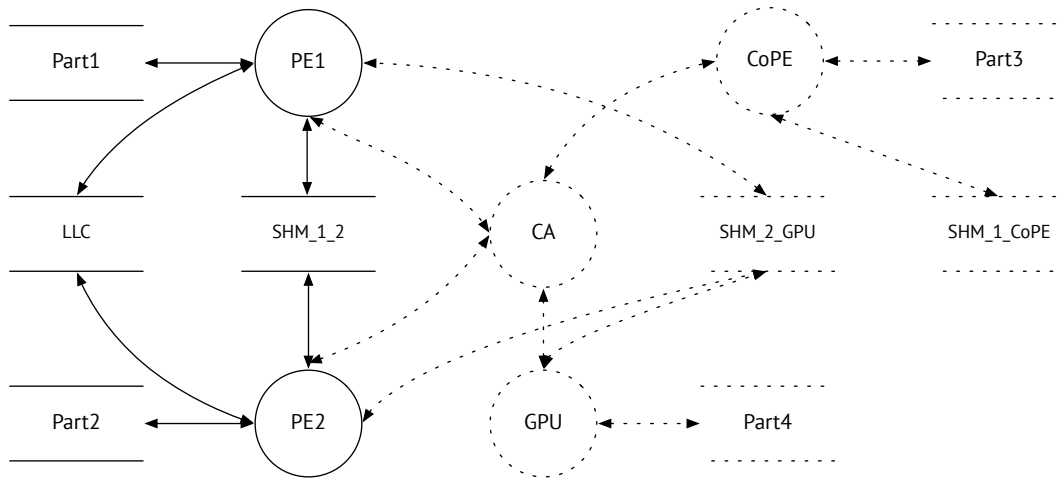


Fig. 3.11 DFD of the technical platform. Dashed items are out-of-scope.

same threats as the considered ones. As a result, the findings are transferable to the rest of the elements. Table 3.2 shows the list of STRIDE-per-element threats which the thread model application reveals.

Table 3.2 Technical DFD: STRIDE-per-element. Check marks in brackets state out-of-scope threats.

#	Element	Threat					
		S	T	R	I	D	E
1	<i>PE1</i>	✓	✓	(✓)	(✓)	✓	✓
2	<i>PE2</i>	✓	✓	(✓)	(✓)	✓	✓
3	<i>Part1</i>		✓		(✓)	✓	
4	<i>Part2</i>		✓		(✓)	✓	
5	<i>LLC</i>		✓		(✓)	✓	
6	<i>Part1 data transfer</i>		✓		(✓)	✓	
7	<i>Part2 data transfer</i>		✓		(✓)	✓	
8	<i>PE1 cache transfer</i>		✓		(✓)	✓	
9	<i>PE2 cache transfer</i>		✓		(✓)	✓	
10	<i>PE1 SHM access</i>		✓		(✓)	✓	
11	<i>PE2 SHM access</i>		✓		(✓)	✓	

Furthermore, Table 3.3 elaborates in detail about the applied threats. In this regard, as is done for the logical threats, the explanation includes practical examples on how such threats could be realized. Also, the violated security goals or missing protection mechanisms are stated. On this level, the combination of threats and protection goal violation will lead to security requirements for each system element.

Notwithstanding, the data flows are mentioned in the DFD and the further investigation focuses on the elements which process or store data. This is motivated by the previously set limitation that the CA is treated as opaque to the system elements. Consequently, the threat analysis omits the technical facilities of data flows.

PE_i is tampered with by adjacent PE_j .

PE can be tampered with in many ways. They contain plenty of facilities which are worth for modifying, including registers, arithmetic, and the caches. Truly, the DFD omits some interactors and interfaces which would be capable of tampering with a PE directly. Debugging interfaces such as JTAG [1] are widespread in embedded MPSoCs and usually capable of interfering with the data even in processor registers. Practically, those opportunities do not apply when considering a modification by an adjacent PE. Consequently, the threat of directly tampering with adjacent PE is not considered any further. However, the modification of data and instructions within the memory partition the PE utilizes is subject to this consideration.

$Part_i$ is tampered with by PE_j

Memory partitions contain a plethora of valuable information for a threat agent. Obviously, all instructions and data structures which facilitate the function implementation are compiled at this place. Therefore, threatening the integrity of memory partitions ($Part_i$) has a major impact to the system behavior and consequently on the system qualities. As a result, the focussed attack for the threat [*MC-domain is tampered with by adjacent MC-domain.*] is tampering with the memory base of which a PE is operating on.

PE_i disrupts PE_j access to LLC.

The proper operation in terms of performance is a key aspect when it comes to the availability of PEs. The dependency on the qualities of the MC-domain is direct. If the PE is degraded in performance, it effects the operation of the functions within the MC-domain executed by the PE. With respect to the DFD (compare 3.11), this

Table 3.3 STRIDE-per-element analysis.

Type	Threat	Description
S	PE_i is tampered with by adjacent PE_j .	A PE is impersonated by another by compromising a communication channel, for example. Communication entities (PEs) are improperly authenticated.
T	$Part_i$ is tampered with by PE_j	PE is tampered with by another by modification of data or control flow. Data at rest or transfer facilities does not implement proper access control mechanisms.
T	PEi tampers with LLC	PE is tampered with by adjacent by modification of data or control flow. Data at rest or transfer facilities does not implement proper access control mechanisms.
T	PE1 tampers with Part2	Part2 is tampered with by PE1 by modification of data or control flow. Data-at-rest (storage) facilities does not implement proper access control mechanisms.
T	PE2 tampers with Part1	Part1 is tampered with by PE2 by modification of data or control flow. Data-at-rest (storage) facilities do not implement proper access control mechanisms.
D	PE1 disrupts the service of PE2 or vice versa.	Mount a DoS attack by over committing shared resources. Resource accesses are not scheduled by a privileged instance.
E	PE1 elevates privilege of PE2 by compromising it's control flow. PE2 elevates privilege of MC1.	PE is compromised with by another due to the exploitation of a vulnerability to elevate/escalate privileges. Improper isolation mechanisms (vertically/horizontally).

correlates with the touch-points (surface) between two PEs, that are represented by the incoming data flows. These touch-points are the *LLC* or *SHM_1_2*. As a result, by disrupting either *LLC* or *SHM_1_2*, the availability of the competing PE is interfered with.

It should be differentiated between compromising LLC and a SHM memory partition. LLC represents a piece of hardware. Practically, there is no opportunity to change anything in the way they are implemented. The SHM partition is technically a portion which is virtually owned by a PE or by multiple PE. The means to access the partitions

relies on the software implementation of a communication protocol. For example, this could be an entire Ethernet communication stack or a simple queue and semaphores for signalling. As a result, the proper function of the SHM relies upon a higher level implementation, rather than the intermediate level. Therefore, the availability of the SHM partition such as the *SHM_1_2* is out-of-scope.

However, the disruption (DoS) of the LLC is crucial to be investigated.

3.3 Risk Analysis

3.3.1 Impact Analysis

According to the previously identified threats, here a quantified impact estimation is shown. In Table 3.4 the analysis results are given. The actual severity values refer to the approach presented in Section 2.2.1. The severities are calculated by considering the impact to the entire vehicle function. This means that the analysed function is regarded in its greater context. The leading question for each threat is: *what is the impact to the vehicle if the function is compromised?*

The presumed stakeholders for this case study are the OEM and the driver (compare Section 3.1.4). Although this is a limited set of stakeholders, they are representative to elaborate on the impacts.

Table 3.4 Impact analysis (**S**afety, **F**inancial, **O**perational, **P**rivacy).

Threat	S	F	O	P	Rationale
MC1					Cluster Instrument
S	2	1	1	0	Safety and operation is affected
T	3	1	2	0	Safety relevant data could be falsified. Vehicle still operational with compromised function.
R	-	-	-	-	Not applicable to scenario
I	0	1	0	1	Reduced reputation of manufacturer leads to financial impact. Disclosed private information of the driver possible.
D	2	1	2	0	Disruption of MC1 effects total operation of the vehicle with safety relevance. For example, tell-tales could be missed.
E	3	1	2	0	MC1 handles vehicle bus communication. Threat of sending unauthorized messages to the bus. Higher impact on safety.
MC2					Infotainment
S	1	1	1	0	Limited impact on stakeholders.
T	1	1	1	0	Vehicle fully operational with compromised function. Chance of issuing a " <i>sound-shock</i> " to distract the driver with a low safety impact conceivable.
R	-	-	-	-	Not applicable to scenario.
I	0	1	0	2	Reduced reputation of manufacturer leads to financial impact. Due to media and connectivity, mid impact on the privacy of the driver or owner.
D	1	1	2	0	Infotainment is a comfort function. Hence, not mandatory for vehicle operation. However, mid annoyance to the driver.
E	2	1	2	0	MC1 handles vehicle bus communication. Threat of sending unauthorized messages to the bus. Higher impact on safety.

In general, the MC1 which represents the cluster instrument implies higher safety severities compared to MC2. This is motivated by the fact that MC1 involve more safety-relevant functions than MC2. These include the display of warnings (tell-tales). Warnings include events such as low tire pressure. The infotainment domain (MC2) handles a comfort function. A likely scenario of an infotainment unit threatening the safety of the driver is for example distraction by deafening audio playback ("*sound-shock*"). Financial impacts are lowly expected on both sides. Due to the necessity of updates, a slight increase in cost is expected at OEM side. Furthermore, compromises always have a negative effect on the reputation. This might result in a reluctant purchase behaviour from customers. The operation of the vehicle is slightly affected by the given threats. In any case, the vehicle will still be operational except if an elevation of privilege is achieved and the system gains the capability to issue other messages on the vehicle than allowed. Privacy concerns play a role mostly in the infotainment domain. Here, privacy-relevant data is processed. These include, for example, contact data in the driver's smartphone or GPS positions of the navigation software.

3.3.2 Attack Potential Analysis

According to the given technical context, the potential attacks given in Section 3.2.1 are now rated for their attack potential.

Table 3.5 Attack: PE s memory base is tampered with by adjacent PE .

Factor	Level	Value
Elapsed time	\leq Week	1
Expertise	Expert	6
Knowledge of system	Restricted	3
Window of opportunity	Unnecessary/unlimited	0
Equipment	Bespoke	7
AP: Moderate		17

Attack: PE_i is tampered with by adjacent PE_j

Rationale

Elapsed time: The investigation and reconnaissance are expected to take approximately one week. This is in accordance to the levels estimated for the other factors.

Expertise: A potential attacker needs to apply expert knowledge to mount the attack. MPSoCs or generally the embedded area deals with highly customized non standardized architectures. The re-engineering of concepts and designs is subject for rigid preparation.

Knowledge of system: Some of the information or documentation of the SoCs is sometimes declared to be non-discloseable to the public (confidential). This implies a higher effort in the re-engineering.

Window of opportunity: This is considered to be unlimited due to the fact that it was assumed that the attacker already found its way into the system.

Equipment: Handling embedded devices is subject to the application of bespoke equipment. This includes programming interfaces such as JTAG debuggers. This equipment has to be built or purchased for a particular ToE.

Table 3.6 Attack: PE_i disrupts PE_j access to LLC .

Factor	Level	Value
Elapsed time	\leq Week	1
Expertise	Expert	6
Knowledge of system	Public	0
Window of opportunity	Unnecessary/unlimited	0
Equipment	Specialized	4
AP: Enhanced basic		11

Attack: PE_i disrupts PE_j access to LLC

Rationale

Elapsed time: The investigation and reconnaissance is expected to take approximately one week. This is in accordance to the levels estimated for the other factors.

Expertise: An potential attacker needs to apply expert knowledge to mount the attack. MPSoCs or general the embedded area deals with highly customized non standardized architectures. The re-engineering of concepts and designs is subject of rigid preparation.

Knowledge of system: It is assumed that the effects of a DoS attack are measurable without non-disclosed information of the system.

Window of opportunity: This is considered to be unlimited due to the fact that it was assumed that the attacker already found its way into the system.

Equipment: DoS is expected to be caused in a state where the device operates normally. The effects of an attack might be observable specialized measuring software.

3.3.3 Risk Definition

According to the security risk level Table 2.4 and the analysis results from the impact and attack potential analysis, here, the risk analysis results are presented in Table 3.7.

Column AP refers to the results from Section 3.3.2. The Attack Likelihood (AL) results from the correlation of the AP and the AL levels given in Table 2.3. The impact severity is split into the C and S component. S is directly derived from the safety severity given Section 3.3.1. The C value represents the maximum severity over the classes financial, operational and privacy with respect to the threat category.

Table 3.7 Risk analysis results.

Threat	Attack	AP	AL	Impact Severity	Risk Level
DoS	PE_i disrupts PE_j access to LLC	11	4	C2 S2	4
Tampering	PE s memory base is tampered with by adjacent PE	17	3	C2 S3	4

3.3.4 Hypothesised Attacks

As a result from the threat and attack analysis and the risk assessment, here the hypothesised attacks are formulated and further investigated in the vulnerability analysis. The conceptual portion of the vulnerability analysis is given in Chapter 4, whereas the penetration test is described the Sections 4.1.2 and 4.2.2.

Hyp-Attack1: PE_i disrupts PE_j access to LLC -> Risk Level 4

Hyp-Attack2: PE s memory base is tampered with by adjacent PE -> Risk Level 4

3.4 Summary

This chapter formed a risk assessment on a ToE that functionally implements a driver information system. The outcome are two hypothesised attacks that are investigated in detail thoroughly the following chapters. As a case study, this ToE facilitates the approach of a mixed-criticality system in an automotive deployment. Functionally, it consolidates two typical automotive functional domains. It includes a cluster instrument domain, which operates a common display and the HMI to the driver, as well as an infotainment domain which mainly operates media and connectivity functions. These two domains run a common MPSoC Platform which implements the AMP system utilization paradigm. A threat context analysis elaborated upon the capabilities of adversaries in the given context. Here, the motivation of an adversary is to control one of the asynchronous domains within the ToE has been discussed. With the threat and attack analysis, the second part of this chapter identified threats from the decomposed system elements. Here the interference with other asynchronous domains by tampering and denial-of-service has been considered in the attack analysis. These results have been discussed in the risk analysis in order to quantify the findings by determining a risk. Here, two hypothesised attacks have been revealed, that will be assessed in further detail in the following chapters. These hypothesised attacks are:

Hyp-Attack1: PE_i disrupts PE_j access to LLC

Hyp-Attack2: PE 's memory base is tampered with by adjacent PE

For both attacks the identified risk level of 4 was identified. The resulting countermeasures are required to increase the AP of the vulnerabilities to the level *High* in order to reach a lower risk level. This means *Hyp-Attack1* is to be increased from *Enhanced Basic* to *High* and *Hyp-Attack2* from *Moderate* to *High*.

Vulnerability Assessment of the Intermediate Layer Attack Surface

Contents

4.1 Hyp-Attack1: PE_i disrupts PE_j access to LLC	99
4.1.1 Vulnerability Analysis	99
4.1.2 Penetration Test: Cache Thrashing	113
4.2 Hyp-Attack2: PE's memory base is tampered with by adjacent PE	120
4.2.1 Vulnerability Analysis	120
4.2.2 Penetration Test: Co-Processor Exploit	125
4.3 Generalized Vulnerability and Exploitation Pattern	131
4.3.1 Modelling Protection Architectures	132
4.3.2 Preliminaries of an AMP system	132
4.3.3 Breach Access Controls on Intermediated Level	133
4.3.4 Denial-of-Service a Shared Resource	135
4.3.5 Identification of Attack Surfaces in AMP Systems	136
4.3.6 Primary and Secondary Assets	138
4.3.7 Attack Objectives and Scenarios	139
4.4 Summary	142

"The essence of hacking is finding unintended or overlooked uses for the laws and properties of a given situation and then applying them in new and inventive ways to solve a problem whatever it may be"

Jon Erickson - Hacking: The Art of Exploitation

Vulnerability assessment as such is an offensive method to determine in which ways a ToE is exploitable. Analysts should think as attackers would do. In other words, they hack their own devices. Hacking is considered to be an art, sometimes referred to as *esoteric engineering* [40]. Truth be told, computer science and esoteric methods obviously don't compare very well in the first place. However, a quite significant security research community contributes to the improvement of the computer system protection, with their work on recent discoveries of vulnerabilities. They optimise attack methods regarding their effectiveness, degrade the performance of countermeasures or just simply circumvent them. Either way, it is about to be one step ahead of adversaries, because this knowledge can be used to build a proper protection strategy.

One might ask, why does it make sense to think about the exploitability of my system? Why do researchers spend effort in that direction and not the other way around? System architectures, particularly protection architectures, are built on assumptions. For example, this is often the anticipated capability of an adversary, particularly when it comes to cryptographic protocols. What hackers mostly do is to break the assumptions on which such systems are built, rather than break a security countermeasure directly [44]. Such countermeasures, a cryptographic protocol, for example again, is publicly reviewed or even formally proven. To aim for breaking the mathematics behind it is a high investment of time and resources. In many cases, it is much easier to leak, for example, the private key material from a communication endpoint.

As a result, vulnerability assessment during the development of a system can be seen as a test of assumptions. This is particularly true for the AMP-paradigms. Hardware was designed having traditional SMP hypervisor paradigms in mind. Even though modern MPSoCs are heterogeneous and highly integrated systems, it might be obvious that if an administrative software layer is moved away to design an AMP system, some security issues would arise.

Within the security cycle, the vulnerability assessment covers the inductive driven phase. It aims to discover specific instances of vulnerabilities and attack instances.

In this chapter, the hypothesised attacks defined in Section 3.3.4 are conceptually analysed. The results on cache thrashing are based on findings of Schnarz et al. in

[144, 146]. This considers the hypothesised attack: *Hyp-Attack1: PE_i disrupts PE_j access to LLC* and is elaborated upon in Section 4.1. Furthermore, the hypothesised attack: *PEs memory base is tampered with by adjacent PE* is examined in Section 4.2. The findings are based on the results of Schnarz et al. [143, 145]. Both misuse cases are evaluated following the concept introduced in Section 2.2.1. In Section 4.3, a generalised analysis of the findings is described.

4.1 Hyp-Attack1: PE_i disrupts PE_j access to LLC

4.1.1 Vulnerability Analysis

The inherent aim of Denial of Service (DoS) attacks is the interference with the performance of the targeted entity. It aims to degrade the performance to a level where particular functions cannot be executed with the intended quality. In practice, this might be the over-commitment of a certain capability of a service. Examples are the handling of messages or requests. The service is forced to operate illicit data and is prevented from fulfilling the original functionality.

Cache thrashing, in general, is a phenomenon which appears due to inappropriate use of a processor's caching infrastructure. Two scenarios lead cache thrashing into taking effect. One is when the cache coherence protocol (such as Modified Exclusive Shared Invalid (MESI)¹) is forced to synchronise data between processors or main memory due to concurrent accesses. The second scenario is when two or more entities (such as processors) exhaust the limited capacity of the shared cache and force the infrastructure to evict and re-fetch data inefficiently. This work is concerned with the last case.

The result of cache thrashing is an increase in latency to access data in memory. In other words, the memory bandwidth and therefore the performance of the system degrades. Each time that entity ($E1$) makes use of the memory data or instructions will be loaded into a particular location of the cache. If another entity ($E2$) concurrently accesses the same location inside the cache the cache management first writes the former values back to main memory to free the space for $E2$. The other way around is also true: if $E1$ regains the focus of the cache $E2$'s data will be evicted. This causes the interference between the two entities.

¹MESI is a protocol to enforce the coherence of data shared in multiprocessor systems. The acronym MESI describes the discrete states a single data can reside in.

Cache thrashing can appear on several system levels. On the application level, for example, two or more concurrent processes or threats could interfere with each other. In this case, both types of cache thrashing are possible whereas the thrashing induced by the coherence protocol at these levels impacts applications which frequently access common data structures. Good examples for this situation are sorting algorithms [104].

Attack Objective and Scenario

From an adversary perspective, the cache thrashing can be induced to mount a DoS attack to a target MC-domain. Particularly in AMP-based MC-systems, the second type of cache thrashing turns into focus. Whereas the cache coherence based thrashing applies for shared data, the eviction of data of concurrent processing elements does not rely on the existence of shared data. Under those circumstances, the adversary just needs to find a particular way to force the data eviction from its target.

For simplicity, the concept is shown concerning the driver information system considering two MC-domains. Each MC-domain runs a private PE, including a private L1 cache. Both PE share a L2-cache of a certain size (compare with Section 3.1.2 for the detailed case study description). Figure 4.1 depicts the considered cache thrashing scenario. It is considered that *MC2* is the targeted domain, thus the victim of the attack. Hence, the attacker who is assumed to reside in *MC1* aims at deliberately decreasing the *MC2* performance to operate on the target frame denoted by *t*. It is assumed that *t* contains data which is critical to the overall mission of the function. In practice, this might be a critical value from a sensor in the vehicle.

Caching Terminology and Design

The general intention of the integration of caches to processors is to speed up access to frequently used data in memory. The memory in computing systems is hierarchically organised [65]. Caches usually consist of several storage levels which contain the data and instructions. A particular kind of cache-logic manages the data inside the cache storage. Typical functions of the cache-logic are the coherence protocol and the write-back strategy. Data in the cache is typically referenced via memory addresses. Two types of cache addresses exist. These include virtual caches, which are sometimes referenced to as logical caches, and physical caches [160]. In physical caches, the requested addresses are translated by a MMU. In logical caches, the data inside the cache is referenced by the virtual addresses. In this work, the scope is on physical caches. In order to reference the data in the cache, the respective requested memory

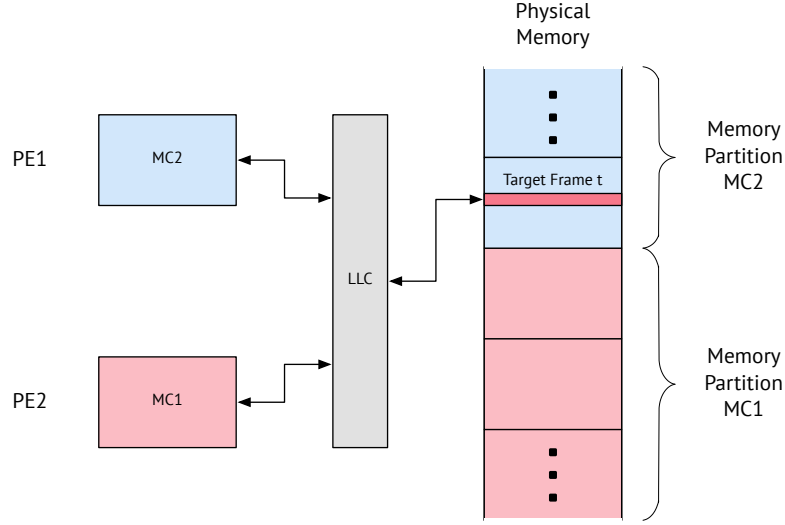


Fig. 4.1 Considered cache thrashing scenario.

address is divided into a *tag* and a *word* section. This is elaborated upon in the later sections.

Despite the highest memory levels, which are the processor's registers, several cache levels can be implemented. Cache levels closest to the processor are commonly referenced to as $L1$. Further levels are denoted as $L2$ or L_n , respectively. The cache level closest to the main memory is commonly referenced to as the LLC. In multi-core systems, some cache levels are private to the processor, and some are shared between multiple processors. The size of caches usually expands from the lower to the higher levels.

Three important parameters of caches are the *cache size*, *cache-line size* and the *associativity* [70]. Aside from the cache size which defines the capacity, the smallest addressable entity within a cache are Cache Line (CL). CL have a fixed CL size, such as, for example, *64 Byte*. Data which are loaded from the main memory into a particular CL are referred to as the Memory Line (ML).

The relationship between the memory and the LLC is referred to as the associativity scheme. It defines the number of possible locations a single ML can be loaded to the cache. Hence, the associativity degree is usually fixed and therefore immutable. The location where a ML is loaded to is denoted as CL_i . The associativity between the LLC and main memory can be fully associative or organised into associativity-cache-way sets. Fully associative means that each CL can be loaded to all CL positions in the LLC. However, due to the sake of efficiency, in the vast majority of LLC implementations,

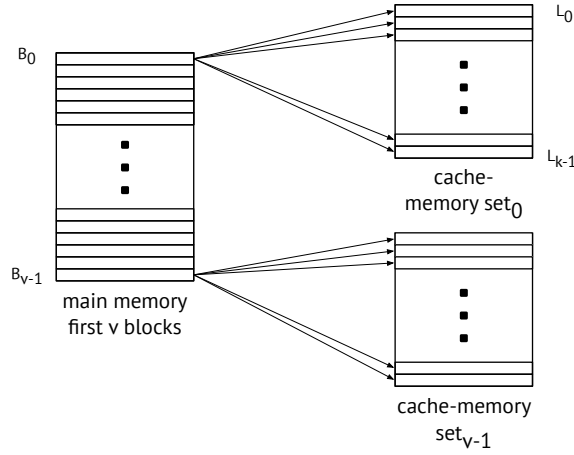


Fig. 4.2 Cache associativity mapping. [160, p. 132]

caches are divided into Way-Sets (WSs) [160]. WSs have a certain size, which are defined by the number of CLs they can contain. If one contains eight CLs, it is referenced as an 8-way-set associative cache. A commonly used associativity for LLCs is 16-way. In way-set caches, a specific ML is always loaded into a specific way-set. This principle is depicted in Figure 4.2. To put it in another way, fully associative caches consist of a single way-set. Referring to Stallings [160] and Hill et al. [70] the following applies for way-set caches.

$$m = v \times k$$

$$i = j \bmod v$$

where

i = way-set number

j = main memory block number

m = number of lines in the cache

v = number of sets

k = number of lines in each set

It depends on the replacement algorithm to determine the specific position within a WS (denoted by CL_i) to which a ML will be loaded. Depending on the specific technical realization, this could be done by a managed Least Recently Used (LRU)

algorithm or upon a randomised scheme. The CL that gets replaced will be written back (evicted) to the main memory. In the literature, the situation where requested data does not reside in the cache is referred to as a cache-miss. The other way around is also true: when the data is existent in the cache a cache-hit occurred. Optimising the cache-hit rates of caches is the target of a wide range of research [70] whereas increasing the cache-miss rate is the target of the DoS-attack. In Table 4.1, the caching key parameters are summarized.

Table 4.1 Caching terminology and parameters.

Sign	Description
WS	Way-Set
CL	Cache Line
ML	Memory Line
MB	Memory Block
CL_{Size}	Size of single cache line
$Cache_{Size}$	Size of cache
ML_{Size}	Size of single memory line equals to CL_{Size}
WS_i	Identifies a specific set in the cache
$Cache_{Assoc.}$	Cache Associativity
CL_i	Identifies a specific CL in the cache
ML_i	Identifies a specific CL in the memory
v	Total number of way-sets in the cache
m	Number of CLs in the cache
k	Number of CLs in each way-set
b	Number of memory blocks in main memory

The number of way-sets, denoted by v , is calculated by:

$$v = \frac{Cache_{Size}}{CL_{Size} * Cache_{Assoc.}}$$

Exploitation of the Cache Associativity

Given the terminology and key parameters of caches an adversary aims at degrading, the computational performance is to increase the cache-miss rate for its victim. In order to achieve this goal, in the following, the cache associativity is examined in more detail. In general, the memory mapping which is applied to the MMU has no relation to the memory association. As a result, if the adversary aims at exploiting the

associativity scheme, they need to flood a particular WS in which the targeted ML resides.

Preliminaries and Threat Capabilities are parameters the attacker has assumed to be known such as the LLC cache size, cache-Line size, associativity degree and the physical address of the targeted memory line. The former parameters are usually part of the technical documentation of the hardware, and therefore it is feasible that adversaries are aware of those parameters. To determine the physical address of the target memory line is out of scope for this consideration. As a result, it is presumed to be aware of the address.

The principle of sharing a k-way-cache by two processing elements working on two different main memory partitions is shown in Figure 4.3. For the sake of simplicity, the private L1 cache is omitted. The example shows PE1 and PE2, each working on two distinct memory lines. Furthermore, this example shows WS0 which is shared by an ML from both memory partitions. In general, this means that each ML belongs to a specific WS in the cache. Such as:

$$ML_i \in MB_n \rightarrow CL_j \in WS_i$$

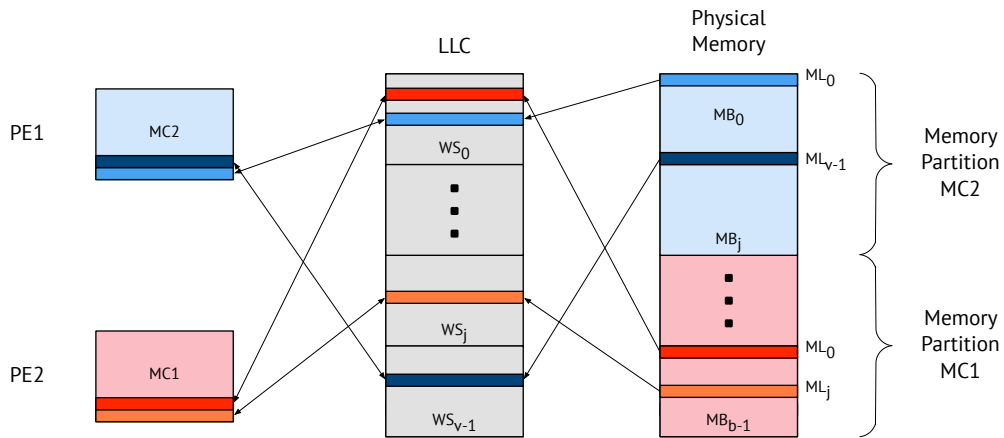


Fig. 4.3 Shared way-set cache principle.

Attack Tree

Here, the subsequent actions to exploit a shared k-way-set are described. Figure 4.4 depicts the attack tree of the DoS attack. The root of the tree states the ultimate goal of the attack, which is to deny the service of a target MC-domain. To increase the cache miss-rate for the target is the method for achieving the goal and is stated in the next level of the attack tree.

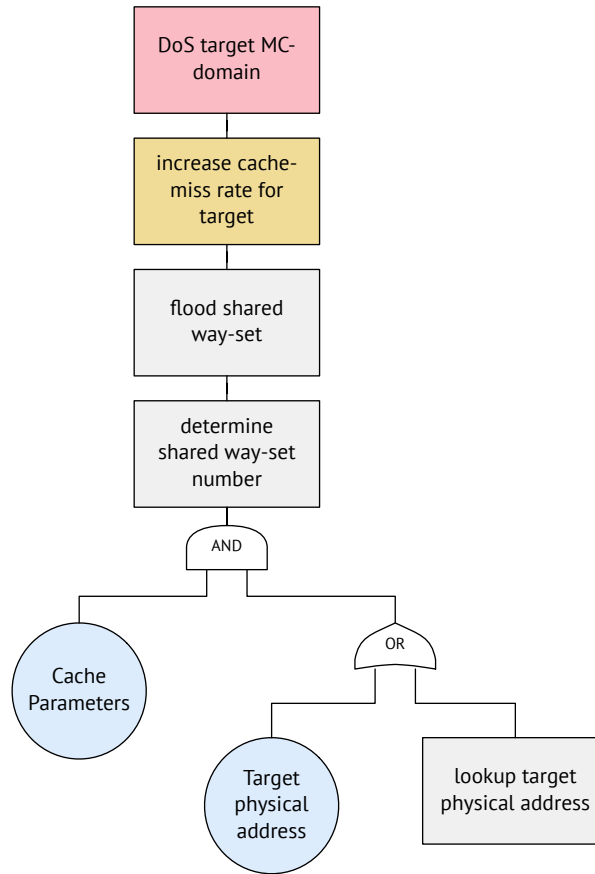


Fig. 4.4 Cache DoS attack tree.

As it was defined that the *cache parameters* previously are presumed to be known to the adversary, they appear as an initial state in the attack tree diagram. Those parameters are usually documented in the technical specification of the applied hardware architecture. More importantly, the *target physical address* is either known by the adversary or looked up in the main memory by any action which is out of scope for this consideration. The core actions are essentially the sub-goals of *determine shared way-set* and *flood shared way-set*. The latter goal aims at producing the actual

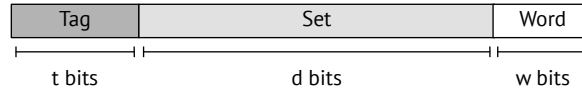
performance impact on the side of the victim whereas the former seeks to set up the environment. In the following sections, these two actions are detailed.

Determination of Target Set (WS_v)

By the reference of the targeted physical address, the target WS can be identified. In the following, this particular WS is denoted by victim WS (WS_v). Substantial preliminaries are the technical properties of the cache. In this case, these include the cache-line size (denoted by CL_{size}) and the WS associativity (denoted by k). Given by those values, WS_v can be determined by the following equation.

$$WS_v = \frac{TargetAddress(PA)}{CL_{size}} \bmod v$$

Alternatively, WS_v can be directly derived from the physical memory address. The memory addresses for k -way-set caches consist of three portions. These components are *tag*, *set*, and *word* [160].



where

$$t = s - d = \text{tag size}$$

$$d = \log v = \text{set size}$$

The d bits specify a particular set within the cache, which is in this case WS_v .

Flood Target Way-Set

Flooding a specific WS causes an increase in cache-misses in that particular WS. As it has been revealed previously, the WSs are the shared memory spaces within the LLC. To introduce a performance impact to target a MC-domain, the adversary just needs to over-commit the particular WS to which the targeted physical address is assigned to. Flooding or over-committing is introduced by frequently fetching cache-lines into the WS in a way that the cache-management is forced to evict other cache-lines. If the attacked PE tries to access its data, it suffers from the higher access latency of the main memory due to a cache-miss. The procedure for forcing this cache-miss and

reload penalty is detailed in the following. Sufficient memory lines need to be allocated and afterwards, accessed very frequently in a loop.

The number of memory lines to be allocated equals the number of cache-lines in the WS (way-set size), which is given by k . An allocation procedure assigns k addresses, which fulfil two aspects. They need to correspond to distinct Memory Blocks (MBs), and all have the same ML_{ID} (offset) within those blocks. This concept is depicted in Figure 4.5. The assigned addresses will be stored in a target address array. According to the *tag*, *set*, and *word* notation, the allocation procedure simply needs to allocate and assign data having an equal t and d bit portion in their addresses.

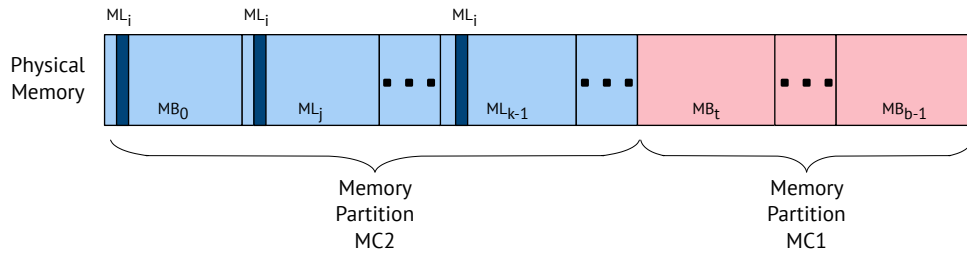


Fig. 4.5 Target address array allocation principle.

The actual flooding is introduced by memory accesses to every entry in the target address array in a loop. Here, the term memory access abstract either read or write operations on external data in main memory. For example, the load-store principle of an ARM RISC architecture those accesses are implemented by instructions such as LDM^2 , STM^3 , $PUSH^4$, POP^5 .

Security Problem Factorization

Timing Consideration In this section, the timing and scheduling issues of memory access are causing the stall of the victim. For the consideration, the time when a cache-line is requested, loaded and evicted is focused. The delta between a request and the actual load of a cache-line is the penalty the PE faces due to the cache-miss. Table 4.2 shows the symbols for the timings in the scheduling diagram.

Figure 4.6 depicts the timing situation of the attack scenario. On the top, the content of the target WS is shown. The bottom part depicts the time-line of the

²LoaD Multiple

³SToRe Multiple

⁴Store data at stack pointer position

⁵Load data from stack pointer position

Table 4.2 Symbols for Cache Scheduling.

Symbol	Parameter	Description
r_i	request time	time at which a cache-line is to be loaded (fetched) into way-set
e_i	eviction time	time at which a cache-line is wrote back to main memory
l_i	load time	time at which the cache-line is loaded into way-set
p_i	miss penalty	time introduced due to cache miss

two competing parties. In the initial state, the WS is filled with cache-lines from the adversary only. This is the flooded state. In this moment (r_{t1}) when the target requests a new cache-line (CL_{t1}), it suffers a penalty (p_1) for the first time due to a cache-miss. Now the CL of the target is resident in the cache. From this state the adversary seeks to force the eviction of CL_{t1} by re-flooding the WS. As a result, on the next request of CL_{t1} the target suffers again a cache-miss penalty p_2 .

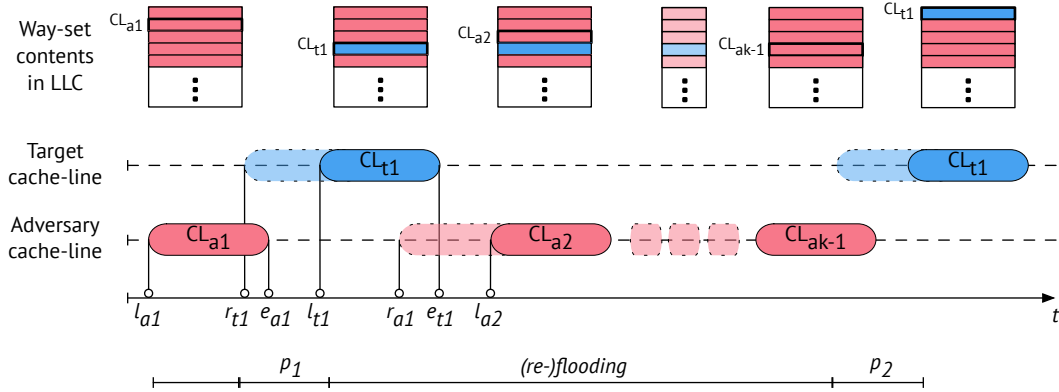


Fig. 4.6 Timing diagram in flooded way-set.

In this scenario, the first access penalty p_1 does not contribute formally to a performance degradation of the target because the cache-line needs to be fetched from main memory anyways. p_2 and potential following cache-line requests contribute to the performance degradation and therefore to the stall of the PE.

Impact of Replacement Algorithms. An important aspect is the algorithm of the cache-management which determines the specific CL to be evicted. The most

commonly used eviction algorithms are LRU and *random*. The latter simply chooses the designated CL randomly. The former tracks which CL was least used in the last period. The eviction algorithm is usually chosen to provide the best effort on average.

In the case of a randomised scheme, the following is considered. It is assumed that the randomization function within the cache-management is non-biased and therefore fair. As a result, the probability of a single CL to be chosen for eviction is $1/k$ where k denotes the number of CLs in the WS. Assuming the adversary controls the entire WS due to its flooding function, the eviction probability of a target CL results in $1/k$. The higher the k-way-set associativity, the lower the probability of a target CL to be evicted.

If a LRU scheme is used, the situation is different. Theoretically, at each point in time it determines which CL will be evicted next. Unfortunately, from the adversary's point of view, this is not visible from the outside, meaning there is no way for the adversary to make a distinction for if he needs to continue to flood the WS to make the target CL least recently used.

As a result, the eviction algorithm plays a secondary role in the implementation of the flooding mechanism. Nevertheless, in randomised schemes, the effect, average penalty over time, is potentially higher when the size of the WSs is smaller.

Dependencies of Access Frequency and Penalty. The impact to t_{access} depends on how often the adversary is capable of introducing the $t_{\delta-DoS}$ penalty. This is directly dependent on the frequency the target consumes data from the cache and memory subsystem. The effect of the DoS attack is higher if the target consumes data in a lower frequency compared to the attacker. With attention to Figure 4.6, the effect is highest if the adversary is capable of flooding the entire WS after the target has accessed its CL.

Metrics to Interfere with Data Access Latency. Increasing the CL miss rate is the goal of the described DoS-attack. In this section, the metrics to quantify this effect are elaborated upon.

Penalty Metric. A commonly applied metric to quantify cache miss-rates or hit-rates respectively is to measure the memory access latencies (denoted by t_{access}). In this Non Uniform Memory Architecture (NUMA), with memory construction, the memory access latencies increase with every cache or memory level (compare Figure 4.7) from the higher level (PE) to the lower level (main memory). Each subsequent level adds a

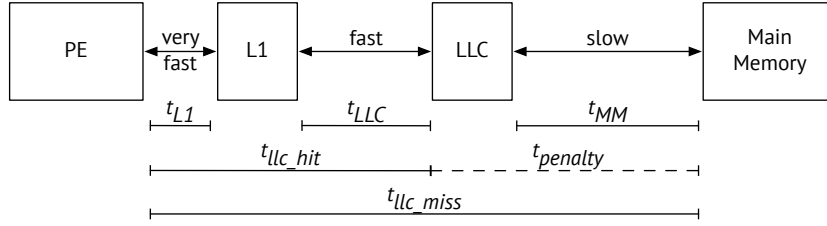


Fig. 4.7 NUMA and cache memory access latencies.

delay to the total access latency. Subsequent latencies are denoted by t_{L1} , t_{Ln} , t_{LLC} and t_{MM} respectively. Usually, the access speeds relate as follows: $t_{MM} > t_{LLC} > t_{Ln} > t_{L1}$.

Table 4.3 Symbols of cache access latency metrics.

Symbol	Parameter	Description
t_i	access latency	time needed to request data
$t_{penalty}$	$\Delta(t_{llc_miss}, t_{llc_hit})$	penalty
t_{llc_miss}	llc miss time	time needed to request a memory-line due to a cache-miss
t_{llc_hit}	llc hit time	time needed to request a cache-line due to a cache-hit

The focused exploitability metric is the delta (denoted by $t_{penalty}$) between a successful LLC cache-hit (denoted by t_{llc_hit}) and a LLC cache-miss (denoted by t_{llc_miss}).

$$t_{penalty} = t_{llc_miss} - t_{llc_hit}$$

From the attacker's point of view, the amount of $t_{penalty}$ is the value they are capable of adding to the t_{access} its target. The portion of the total latency denotes the delay until a PE can consume the requested data. Subsequent latencies between different cache levels and the main memory are indicated by t_{LLC} and t_{MM} respectively.

Penalty Impact on Execution Time. In the previous section, it is elaborated upon as to why the cache-miss penalty is a factor for exploitability. Furthermore, it is shown how the impact theoretically works. Programs usually take a certain amount of time to execute a particular operation, task or algorithm. One factor to measure the quality of this performance is the Worst Case Execution Time (WCET). If the

WCET factor is violated, other dependability issues could arise. Particularly in the automotive scenario, those are safety impacts.

Substantial impacts to the performance of the target are to be expected if there are sequences of memory accesses penalised by a cache-miss. In such cases, the penalties would summarise and result in a tremendous impact on execution time. The severity of the impact is dependent on the actual program. For example, the most crucial aspect is the frequency of the memory accesses. As a result, it is proposed to measure the impact and therefore the exploitability with a mean value of a sequence of memory accesses.

Vulnerability Score

In this section, the CVSS of the identified hypothesised attacks are rated and justified. The Base Score is rated at 5.3 (Medium) according to the metrics given in Table 4.4. The nature of the vulnerability is the degradation of the performance of memory accesses. Accordingly, it impacts only the availability of adjacent domains.

Table 4.4 CVSS base score of the cache-thrashing exploit concept.

Base Score Type	Rating	Score
Attack Vector	Local	
Attack Complexity	High	
Privileges Required	High	
User Interaction	None	
Scope	Changed	
Confidentiality	None	
Integrity	None	
Availability	High	
Base Score	Medium	5.3

Rationale.

Attack Vector. The given exploit relies on local access to the device (compare with the attack tree in Section 4.1.1). Since it is assumed that the attack takes place at the intermediate level, requiring further effort to reach that system level, the severity is set to *Local*.

Attack Complexity. The attacker must prepare the target environment to implement a successful exploit. Furthermore, since it is anticipated that the attack is to be mounted from the intermediate level, an attacker needs to find further vulnerabilities to be successful (compare with the attack tree in Section 4.1.1). Additionally, the knowledge of the cache parameters are preliminarily required for the attack. This has to be revealed by a device specific research or reconnaissance. For those reasons, the attack complexity is scored at *High*.

Privileges Required. The level of privileges an attacker must possess before successfully exploiting the vulnerability is *High*. This is in line with the previous scores since it is assumed that the attack has to be mounted at a privileged level. However, the exposure to higher levels is subject to future work.

User Interaction. The vulnerable system can be exploited without interaction from any user. Therefore, the score *None* is justified.

Scope. The vulnerability affects resources beyond the domain separation of the vulnerable component. In this case, this is the contention of particular WSs. In this case, the score *Changed* is chosen. However, in the case of a shared cache, one might argue that the cache management is a higher privileged entity which fails to arbitrate to competing accesses. Since the k-way-set associative cache is only administered by the replacement algorithm and the memory association, the assumption of dealing with two distinct authorities (the asynchronous domains) is reasonable.

Confidentiality. The confidentiality of information is not affected.

Integrity. The integrity of information or code is not affected.

Availability. The potential impact on the availability has been rated at *High*. This refers to the adversary's ability to introduce a penalty ($t_{penalty}$ compare with Section 4.1.1) to the cache access latency by deliberately interfering with the cache scheduling as it is shown in Section 4.1.1. A rebuttal of the general efficiency of the attack is a cache replacement algorithm and the access frequency that cannot be controlled by the adversary. However, these aspects affect the impact of the mounted attack.

Nonetheless, the granularity of the CVSS metric is rather coarse. From case to case the rating shall be in between *Low* and *High*. In this case, the worst case

impact severity has been chosen. However, this is modifiable with the corresponding environmental vulnerability score, considered in Section 6.1.2 of the security evaluation. For example, taking the targeted context into account, which is the automotive sector, severe functions could be affected by the loss or dramatic reduction of the availability. The attacker can disrupt the availability of the vehicular functionality. The loss of availability presents a direct, serious consequence to the impacted component. However, the attack is only possible for punctuated memory accesses. For that reason, there is reduced performance or interruptions in resource availability but not for the full range of service provided by the affected domain.

4.1.2 Penetration Test: Cache Thrashing

In this section, the penetration test of the cache thrashing concept is described. The test relates to the hypothesised attack shown in Section 4.1.

Experiment Setup

Generally, the experiment aims at demonstrating the cache access latency penalty. This will be approached by measuring the delta between the execution time of a test-program with and without the applied attack. The resulting $t_{penalty}$ is used as evidence to test the hypothesis as stated below.

The hypothesis (**Hypothesis_{e1}**) considers an offensive aspect. The alternative hypothesis considers a significant delta between the results of the thrashed and the non-thrashed measurements. In order to define a reasonable threshold, results from similar attacks have been taken into account. For example, Kim et al. [95] show that through unfair cache sharing on application level the performance of a memory consuming function can be degraded up to 63 percent. However, the test setup and the experimental platform differ from the one used herein. Hence, finding suitable reference values requires at least a setup with similar cache parameters to be comparable. Therefore, here, the effect of the penetration is measured by comparing the mean execution time of two measure runs, which will be introduced in the following paragraphs. In order to test whether or not a significant performance impact is measurable by applying the cache thrashing, the average execution time must be at least higher than normal duty workload plus the standard deviation. The symbols are explained in 4.5. The intention is to prevent from interpreting jitter (variant execution times) as a success of the test.

Table 4.5 Measurement symbols.

Name	Equation	Description
s_i	$s_i \in \mathbb{N}$	CPU cycle count value at the start of iteration.
e_i	$e_i \in \mathbb{N}$	CPU cycle count value at the end of iteration.
k	$k \in \mathbb{N}$	Number of iterations
δ_t	$\left(\frac{\sum_{i=0}^k e_1(i) - s_1(i)}{k} \right) i, k \in \mathbb{N}$	Delta of DoS/thrashing measurements
δ_n	$\left(\frac{\sum_{i=0}^k e_1(i) - s_1(i)}{k} \right) i, k \in \mathbb{N}$	Delta of normal duty measurements
δ_i	$\delta_t - \delta_n$	Delta reflecting identity mapping
s_n	-	Standard Deviation of δ_n
s_t	-	Standard Deviation of δ_n

Hypothesis_{e1}: The adversary domain is able to decrease the computing cache access performance by the deliberate thrashing of a common cache WS.

H₀: The impact on the performance is not measurable ($\delta_t \leq s_n$).

H₁: There is a significant delta ($\delta_t > s_n$) in the performance.

However, the significance of the impact is dependent on the particular use-case of the attacked program. If there are requirements on the worst case execution time (WCET), performance degradations would impact different delta levels.

In Table 4.5, the key measurement symbols are described.

Measurement Setup. To conduct the experiment, a certain set of measurements need to be executed. This paragraph elaborates on the details of the experimental sequences. In general, the setup is as follows: the victim-domain logs the latency of its memory accesses by consuming data out of the target WS in the cache. The adversary-domain floods this WS at the same time.

Cycle Counts. In order to achieve precise measurement results, an appropriate timing reference must be chosen. In this case, CPU cycle counts are used to measure the latencies of memory accesses. Cycle counts enable a fine-grained resolution of relative measure points. ARM processors usually implement a cycle count [10] register. The ARM Cortex A15 processor includes a system control coprocessor (CP15) which

controls and provides status information. One function of this CP15 is the Performance Monitor Control Cycle Counter (PMCCNTR). This PMCCNTR is readable from privileged code by the following assembly instruction:

```
1 asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t": "=r" (result)::);
```

Measure Functions. The experiment involves both MC-domains each of which executes a particular set of functions. The functions are briefly introduced in the following:

Get Time. The `get_time()` function retrieves the cycle count from the PMCCNTR. This is done according to the assembly instruction given in Paragraph 4.1.2.

Prepare Cache Lines. The `prepare_cachelines()` function determines the ML associated with the targeted WS in the physical memory. This ML structure will be iterated by the measurement and DoS loop functions. The structure is built according to the concept given in Section 4.1.1.

Iterate Memory Lines. The `get_next_CL()` function iterates and retrieves the next CL from ML structure.

Modify Cache Line. The `modify_CL()` accesses the current ML and to force the cache to evict another CL from the target WS.

Measure Loop. The `measure_loop()` function iterates over the structure which has been created in the `prepare_cachelines()` function. During each iteration it loads the cycle count timestamp from the PMCCNTR register and stores it in a stack variable. Refer to Listing 4.1, where the measure loop function is presented in pseudo code. The listing shows a constant `TEST_ITERATIONS` which defines how often the loop will run and measure. In the controlled environment of this laboratory experiment, the constant has been set to 100.000 iterations. Since there are no other tasks running on the system, there are no statistical outliers expected which could be caused by a concurrent process.

DoS Loop. Compares to the previous function, however without the time measurements. Refer to Listing 4.2, for a pseudo code example.


```

1 measure_loop(){
2     for (k = 0; k < TEST_ITERATIONS; k++) {
3         reset_counter();
4         ML_structure[] = prepare_cachelines();
5         for (l = 0; l < ML_structure[].end() ; l++) {
6             s = get_cycle_count();
7             current_CL = get_next_CL(l);
8             modify_CL(current_CL);
9             e = get_cycle_count();
10            delta += e - s;
11        }
12    }
13 }

```

Listing 4.1 Pseudo code to accumulate the memory access latencies measured in CPU cycles

```

1 DoS_loop(){
2     for (k = 0; k < TEST_ITERATIONS; k++) {
3         ML_structure[] = prepare_cachelines();
4         for (l = 0; l < ML_structure[].end() ; l++) {
5             current_CL = get_next_CL(l);
6             modify_CL(current_CL);
7         }
8     }
9 }

```

Listing 4.2 Pseudo code to thrash the common way-set.

Measurement Sequence. Technically, the functions are invoked according to the sequence diagram shown in Figure 4.8. After the initial boot sequence (compare with Section 3.1.3) is executed, each of the PE load and execute a Linux Kernel module which contains the testing functions. *PE1* which is considered to be the victim domain first starts a measure-loop and stores its results. These results are the reference values (δ_n) for determining the delta later on. Next, *PE1* triggers *PE2* to invoke the DoS-loop and starts the measure-loop right after. The trigger is used to synchronise the execution of both loops. Technically, the trigger is implemented with the means of a *semaphore* in a shared memory area. After each of the PEs finished the loop iterations, *PE1* stores the results (δ_t).

Measurement Results

Way-Set Occupancy The first measurement determines how the thrashing impact compares to the number of CLs iterated in a single WS. In this case, both systems run the *measure-loop* and measure the latency concurrently. The results to test the DoS method are depicted in Figure 4.9. The most significant impact on the execution performance of the victim-domain peaks at about 457 percent. This means that in

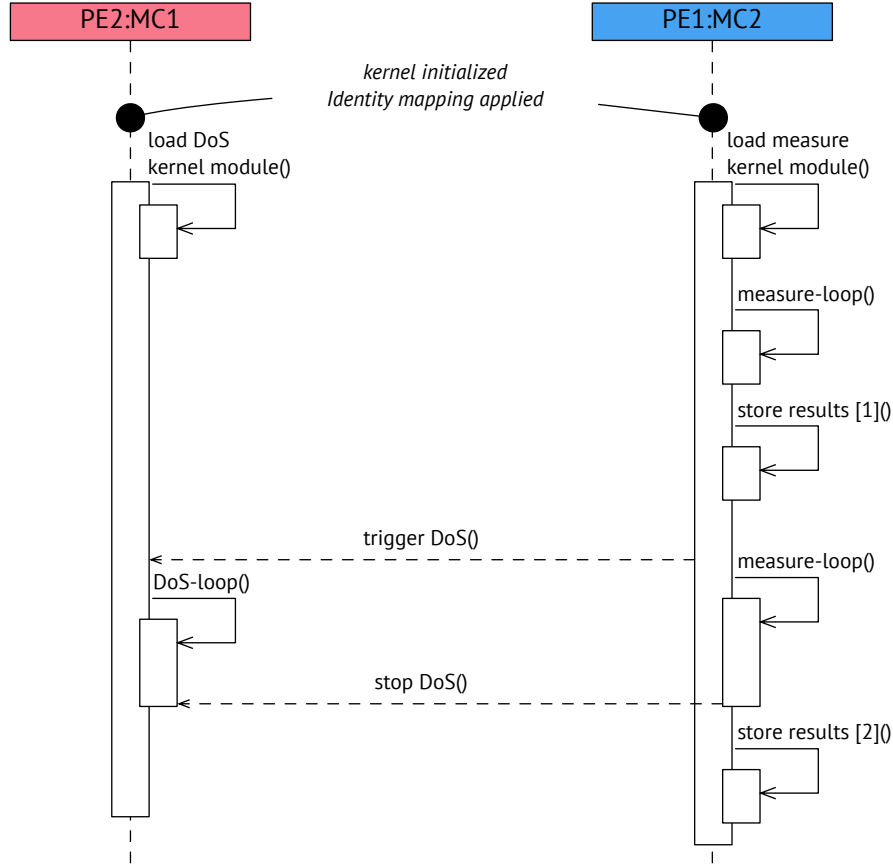


Fig. 4.8 Sequence diagram of thrashing measurements.

this scenario, the consumption memory is delayed by this magnitude. The number of cache-lines which are iterated within the `measure-loop` and `DoS-loop` influence the impact on the delta. Since the experimental platform implements a random cache-line replacement strategy, the probability of a CL eviction is highest when every cache-line in the WS is iterated. According to Figure 4.9, the highest peak of interference appears when the victim and the attacker are using 16 CL.

Table 4.6 Overview of the measurement results.

Mapping	Impact (%)	CPU Cycles (δ)	Std. Deviation (s)
identity	-	3,602 (δ_n)	0,022 (s_n)
identity DoS	3686,90%	132,803 (δ_t)	2,265 (s_t)

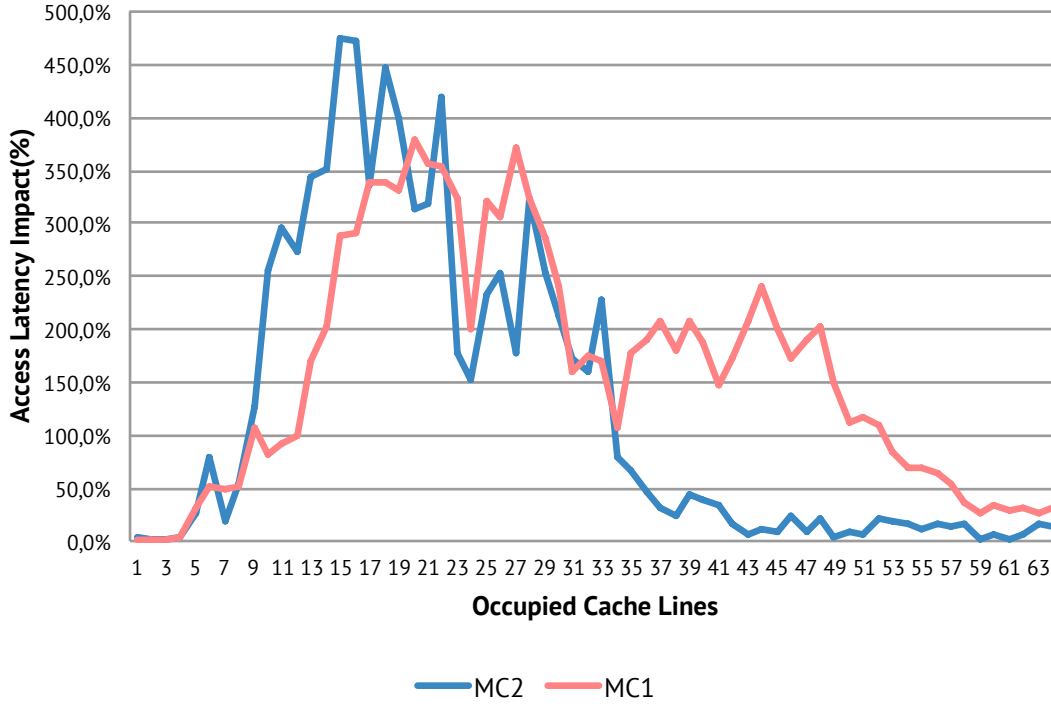


Fig. 4.9 Comparison of way-set occupancy.

Identity Mapping The following results aim at providing evidence to test *Hypothesis_{e1}*. For this purpose, the measurements are conducted as stated in the sequence diagram depicted in Figure 4.8. The victim-domain first executes the *measure-loop* in order to create a reference value. Right after that, the adversary-domain is triggered to invoke the *DoS-loop*. For these measurements, both systems iterate over 16 cache-lines, which produces the highest impact (compare Figure 4.9). The results are shown in Figure 6.1, whereas Table 6.4 gives and summarizes the measured results.

The first *measurement-loop* run reveals the mean cycles count to consume the data within the cache. The arithmetic mean of δ_n is about *3,602 cycles*, which means around 4 cycles are consumed to access the data in the cache. The second set of results reveals δ_t with about *132,803 cycles* (~133 cycles). With these results, δ_i is about

129,501 cycles. This implies that the invocation of the *DoS-loop* in parallel with the *measurement-loop* impacts the mean access cycles at about 3686%. By comparing the s_n and δ_t , the significant increase is clearly observable. As a result, the measurements prove evidence that H_1 applies. Invoking the cache-thrashing is a significant delta in the performance of the memory accesses.

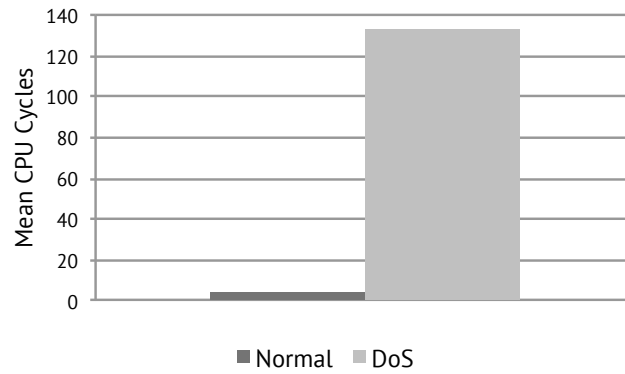


Fig. 4.10 Identity memory mapping.

Temporal Vulnerability Score

The temporal vulnerability score considered in this section refers to the base score given in Table 4.4. Accordingly, the corresponding Base Score is 5.3 (Medium). The pentest is rated at 4.8, which is still rated as (Medium) severity. As is shown in Table 4.7, the temporal vulnerability score manipulates in a positive direction and limits the severity.

Table 4.7 CVSS temporal score of the LLC disruption pentest.

Temporal Metric	Rating	Multiplier
Exploit code maturity	Proof-of-Concept	0.94
Remediation level	Unavailable	1
Report confidence	Reasonable	0.96
Temporal Score	Medium	4.8

Rationale

Exploit Code Maturity In this situation, the score *Functional* or *Proof-of-Concept* is considered. Other metric values, such as *High* do not apply, for example, since there is no autonomous code made available. In this case, *Proof-of-Concept* is applied. During the penetration test, code has been produced which is capable of testing a hypothesis⁶. This code has not been made public. However, the technique and the conceptual consideration is practical to similar systems. Since the exploit code is not runnable with interaction and manual work of the tester, it is reasonable not to apply the metric value *Functional*.

Remediation Level The metric value *Unavailable* is motivated by the fact that no solution is applicable at the given system level. The remediation is subject to the proposed countermeasure.

Report Confidence To the date of discovering the vulnerability, no evidence indicated such impacts to the availability of AMP-based systems. However, the root-cause of this exploit technique is based on well-known issues in real-time computing. Therefore, the metric value is set to *Reasonable* rather than *Unknown*.

4.2 Hyp-Attack2: *PE*'s memory base is tampered with by adjacent *PE*

4.2.1 Vulnerability Analysis

In this section one of the initial hypotheses is conceptually analysed. The question is: *are AMP-based systems prone to DMA attacks?* The answer seeks to deliver an instance for the misuse case that is elaborated upon in the case study (refer to Section 3.3.4. *Hyp-Attack2: PE's memory base is tampered with by adjacent PE*). For that purpose, the technical hardware architecture vulnerability is examined.

DMA-based attacks are often referenced in research according to commodity desktop or server systems, which commonly implement the x86 architecture (compare for example [17, 142, 188]). Hence, in AMP-systems this becomes in focus because of two essential properties of DMA devices. DMA was introduced to detach a general-purpose processor from the burden of memory transfers, in order to handle them asynchronously. In this case, the DMA-controller copies data from one main memory location to another. The memory is transferred within the same address space. DMA-capable devices are

⁶Compare Section 4.1.2

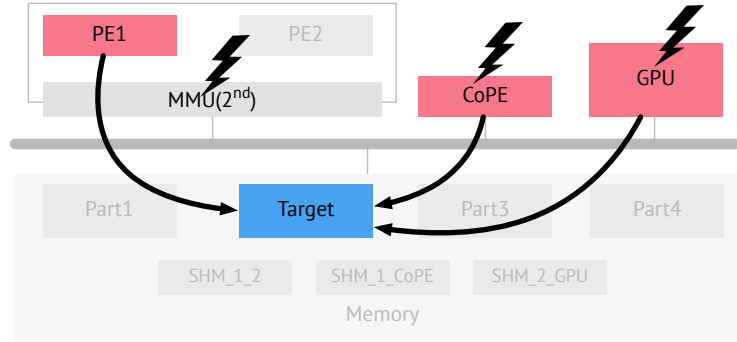


Fig. 4.11 Breach or circumvent memory protection controls in a SoC.

also utilized to copy data between two disjointed address spaces. Practically, this is the case if a subsystem needs to copy data from the system physical address space to a domain partition.

In MPSoC environments, the direct access to memory is an essential functionality in order to integrate the large set of heterogeneous hardware elements. This is particularly true for PEs. The inter-PE communication is often implemented via shared memory communication (Inter Process Communication (IPC)). Thus, each PE needs to implement a memory access capability. In the following, there are two approaches of misusing the memory access capability of co-processors and GPUs. In the later sections, the generalized issue of DMA in AMP-systems is examined.

Co-Processor Exploit Scenario

Roughly, in computation setups, the PE can be divided into two classes. There are elements handling the main function of the embedded device, and there are PE supporting the execution of those main functions. PE that work on behalf of the main function are referred to as co-processors or CoPE. From the functional point of view, these CoPE are working in a master-slave setup. The master PE has the control over the slave PE.

However, from the technical point of view, the CoPEs are implemented as a fully qualified processor running its software stack or firmware. As it is mentioned previously, it also often implements an interface to the CA which is the DMA capability. On the one hand, this is necessary to provide external RAM to the CoPE. The CoPE might also implement private ROM and RAM to execute its firmware. On the other hand, the access to the CA and accordingly, to the main memory (RAM) are needed for IPC purposes.

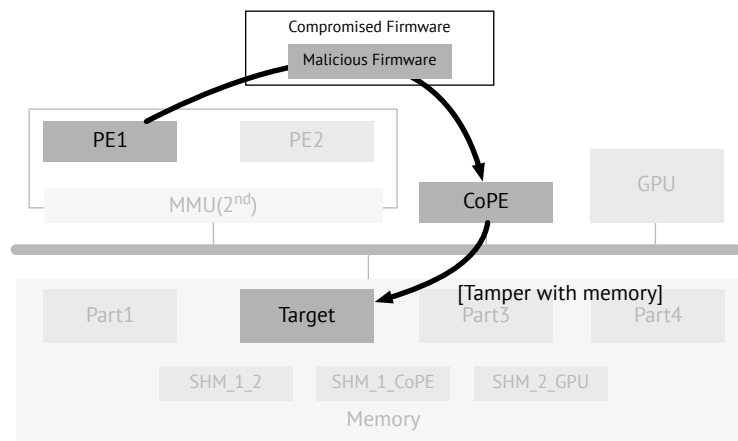


Fig. 4.12 Misuse scenario utilizing a co-processor to attack a target memory area.

That means, as a result, functionally, the CoPE is under control of the master PE, but technically they are independent. This circumstance is adopted in this misuse case.

It is assumed that the master PE implements a proper access control enforcement by the MMU. This is actually, what the attacker wants to breach. The master PE takes advantage of the CoPE in order to let it attack the target on its behalf. The concept is visualized in Figure 4.12. The concept proposes injecting the malicious behaviour by code/firmware that is to be loaded and executed by the co-PE. In the penetration test, this will be demonstrated on commodity MPSoC hardware. The penetration test is described in Section 4.2.2.

GPU Exploit Scenario

In the range of co-processors, the GPU is considerably one of the most powerful forms for the performance perspective. GPUs are not only used in multimedia appliances to visualize content, but also applications requiring a tremendous amount of acceleration in any stream-processing related purpose. In vehicular environments, this might be the analysis of camera data or any artificial intelligence task. As a result, GPUs are widely used and therefore, integrated into MPSoCs.

The key concept of misusing the GPU to circumvent access control mechanisms is similar to the co-processor approach. GPUs in MPSoC environments access the main memory in order to execute the computation on data that resides there. In reference to the case study, the pixel data is efficiently communicated/handled in a SHM memory area.

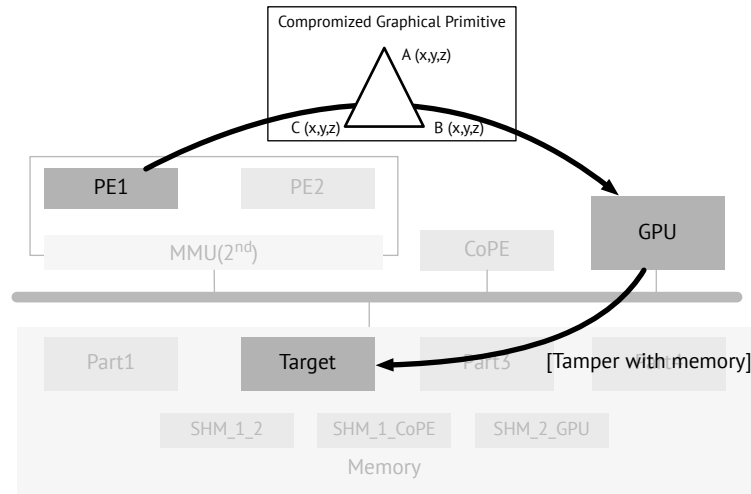


Fig. 4.13 Misuse scenario utilising a GPU to attack a target memory area.

However, in the case of communicating commands to the GPU, the master PE must setup graphical primitives. Those primitives contain the description/construction of the graphics data and the graphics commands to be applied to the graphics data. Particularly, in the description of these graphical primitives is described where the input data resides in memory and where the output should be written. By manipulating these input and output descriptors, the GPU made up the ability to write arbitrary data from one memory location to another.

Security Problem Statement

The previously given exploit scenarios base on the problem that DMA capable system elements might implement an inconsistent access control architecture. That means, on the intermediate level an EoP to accesses to a memory resource cannot be prevented. A detailed analysis of this issue is given in Section 4.3.

Vulnerability Score

Table 4.8 CVSS base score of the access control breach.

Base Score Metric	Rating	Score
Attack Vector	Local	
Attack Complexity	Low	
Privileges Required	High	
User Interaction	None	
Scope	Changed	
Confidentiality	High	
Integrity	High	
Availability	Low	
Base Score	High	8.1

Rationale. With respect to the metrics in shown in Section 2.2.1, the following aspects refer to the exploitability.

Attack Vector. The given exploit relies on local access to the device since it is already assumed that the attack takes place at the intermediate level requiring further effort to reach that system level. As a result, the severity is set to *Local*.

Attack Complexity. The required entry attack vectors and required privileges are discussed in the previous section. With regards to the complexity of the attack, the attacker does not need to conduct a target-specific reconnaissance. The system configuration is considered to be static due to the fixed memory mapping. There is no intended variation from target to target. As a result, once an adversary is able to breach the memory protection for one system, the concept is applicable to other systems in the product-line or vehicle fleet.

Privileges Required. The level of privileges an attacker must possess before successfully exploiting the vulnerability is *High*. This is in line with the previous scores since it is assumed that the attack has to be mounted at a privileged level. However, the exposure to higher levels is subject to future work.

User Interaction. The vulnerable system can be exploited without interaction from any user. Therefore, the score *None* is justified.

Scope. Due to its nature, memory protection breaches allow a wide range of impacts on security goals. In this particular case, the adversary is able to conduct control flow integrity attacks and elevate the privileges of certain functions of the system. This implies a change of scope [61], meaning the attacker is not only able to control the targeted function but also further assets from this position.

Confidentiality. The confidentiality is exploitable by the shown vulnerability. Due to the breached memory protection, an adversary is able to disclose information from the ToE. Accordingly, the impact to the confidentiality is scored at *High*.

Integrity. Compromising the integrity of the ToE is the direct result of an exploit of the breached memory protection. By tampering with the memory within the ToE, a proper function cannot be guaranteed. Therefore, the impact is rated at *High*.

Availability. The availability is not directly endangered by the vulnerability. However, due to the change-of-scope (compare with Section 4.2.1), denial-of-service attacks might be conducted by further exploitation. Accordingly, the availability impact is rated at *Low*.

4.2.2 Penetration Test: Co-Processor Exploit

The heretofore described penetration test implementation realizes the concept breaching memory protections, which is introduced in Section 4.2.1. This contribution is based on the publications shown by Schnarz et al. [143, 145].

Hypothesis_{d1}: The adversary domain (MC2) can utilize the IPU write to a target memory area that belongs to MC1.

H₀: There is no data that can be written into the target memory area that belongs to MC1.

H₁: The data in the target memory area is changed on behalf of MC2.

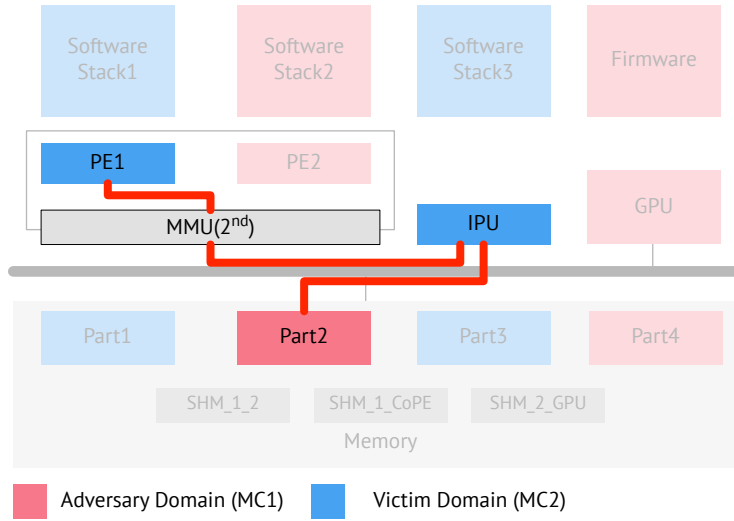


Fig. 4.14 Implemented attack vector.

Attack Vector Implementation

The attack vector utilises the MPU and IPU of the experimental platform⁷ (EVM). The MPU facilitates a double-core PE, each of which executes a MC-domain (compare Section 3.1.2). Furthermore, the IPU is utilised by one of the MC-domains to circumvent the memory protection of the MPU.

Accordingly, the cluster domain (*MC2*) is declared as a victim domain and the infotainment domain (*MC1*) as the adverse domain. In this scenario, it is assumed that the adverse domain has access to the IPU. The assumption is reasonable because of image processing acceleration, which the IPU was intended for, and is a feature utilised by infotainment applications.

To circumvent the isolation, in this scenario an adversary utilises the IPU and exploits its capability to access the main memory independent of the MPU. Since the evaluated architecture of the OMAP5432, does not feature VE or Intellectual Property Memory Management Unit (IPMMU) for the Cortex-M4 processors, the IP-core is capable of accessing the full address range of the main memory. Only the memory-mapping of the IPU has to be manipulated to gain access to the entire memory range. Accordingly, the attack approach is based on injecting a malicious firmware to the co-processor. If the firmware is executed, it compromises the memory-mapping of the IPU's translation table. This results in the capability to circumvent the MC-domain isolation on behalf of the adversary-domain. Figure 4.14 visualizes the attack vector.

⁷compare with Section 3.1.3

Experimental Setup

The experiment is assumed to be mounted during normal system runtime. That means, when the original boot sequence is done, and the OS of both domains are initialized.

Therefore, the experiment is crafted by involving the three system components.

Adversary (MC1) General system setup and installation of the exploitation firmware. Executed by the MPU-PE2 on behalf of the adversary-domain.

Adversary (MC1) Exploit-firmware execution. Executed by the IPU on behalf of the adversary-domain (MC2).

Victim (MC2) Evaluation of success. Observing the target memory area. Executed by MPU-PE1 on behalf of the victim-domain (MC2).

Preliminary to the attack sequence, a target memory area with the address $addr_{target}$ within the victim's memory space is defined. Furthermore, a signature sig will be negotiated between the two domains before the attack. In this example, the string $0xEEAADCC$ is used. If the victim-domain recognises sig at the given address $addr_{target}$, the experiment gained evidence that the hypothesis could be tested.

Attack Sequence

A substantial prerequisite is the initialization of the IPU. This is done by the adversary-domain. Therefore, all of the necessary configuration-registers need to be mapped to the MPU's memory space. These configuration registers include, amongst others, the interfaces to set up the device's internals such as system clock and processor frequencies as well as the setting of necessary base registers that the IPU can access external resources. Most importantly, this includes the L2-MMU-Translation-Table-Base-Register in order to initially provide a well-formed translation-table to the IPU's internal MMU. An overview of registers and their addresses is depicted in Figure 4.15.

Once, the IPU is set up, and the system is running, the adversary crafts the exploitation firmware and provides it with the $addr_{target}$ and sig . In Section 4.2.2, the exploitation firmware is elaborated upon in more detail. In the end, the adversary populates the previously prepared exploitation firmware to where the main memory is so the IPU can execute it. This is referred to as firmware load address (FLA) and denoted by $addr_{FLA}$.

In the next step, the exploit-code that resides within the exploitation firmware is triggered. The attack code attempts to add the malicious MMU entry to the L2-MMU-Translation-Table of the IPU. Finally, the exploit-code simply writes the sig to the

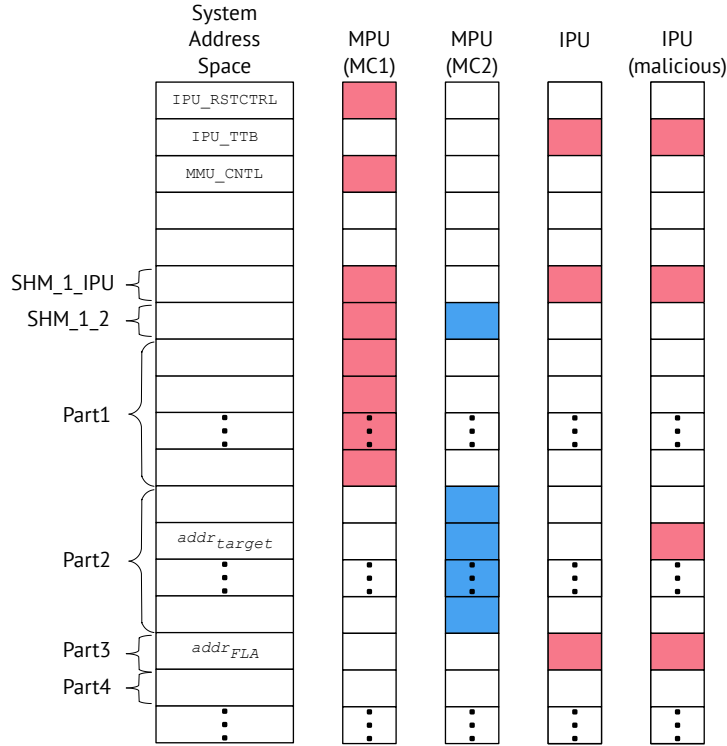


Fig. 4.15 Memory mapping (extract) for the attack vector.

previous written mapped memory area in which $addr_{target}$ resides. A detailed view of the memory mapping is given in Figure 4.15.

The victim's side cyclically checks whether the signature was written to the target area.

Memory Mapping. The IPU processor is connected to the system bus (CA) (L3-Interconnect) through a L2-MMU. The L2 MMU translates from the IPU's VA to the PA of the system. The configuration of the MMU can be done from one of the Cortex-M4 cores or a system bus slave port. By utilizing the MMU control register IPU_MMU_CNTL the IPU MMU is configurable. In the shown example, the MMU is programmed from the IPU locally. Therefore, the control register is mapped to the IPU (compare with Figure 4.15). The IPU's translation table consists of 32 entries, each of which points to a 1MB physical address area [79]. In this PoC-implementation, all entries are mapped to the address where the exploitation firmware is located in the main memory. Since the exploitation firmware size is smaller than 1MB, only the first entry in the translation table is necessary to provide a correct MMU setup. This is the minimally required setup to build a proper virtual address space environment

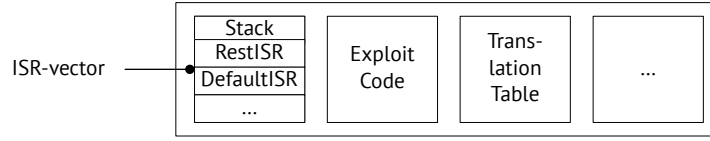


Fig. 4.16 Exploitation firmware structure overview.

for the IPU. Within the exploitation firmware, the very last entry of the translation table is used to point to $addr_{target}$ in the victim-domain. When the IPU accesses the appropriate VA, the M4 is now able to write the signature (*sig*) to the target address.

Exploitation Firmware. As the utilisation of the IPU includes the definition of entry points, interrupt service routines and further configurations, is a common format to craft the exploitation firmware that has been chosen. Therefore, the adversary-firmware is organised in the Executable and Linking Format (ELF) format [31]), which is the common standard for executables in UNIX-like OSs. Furthermore, it is a common technique to craft the binary format of co-processor firmware by utilising ELF. For example, utilization-frameworks such as the *remoteproc* [91] rely on ELF binaries. Figure 4.17 visualises the concept of the exploitation firmware.

According to the ELF specification [31]), the firmware implements the basic code sections to execute commands on a processor. These code sections include the interrupt service routine vector table (*ISR_vector*), a translation table for the *MMU* and additionally the exploit-code section. Unlike the usual usage of ELF-binaries, the reset Interrupt Service Routine (ISR) (*ResetISR*) is utilized as the entry point for the execution. This means that, after the interrupt is raised to the IPU, the *ResetISR* immediately redirects to the exploit-code which reconfigures the *MMU*. Thus, a *ResetISR*-function within the *ISR_vector* of the firmware is registered. This enables the adversary-domain to trigger this function by setting the appropriate value to the reset register. Once the *IPU_RSTCTRL* is raised it delegates to the function within which the exploit will be executed. Furthermore the firmware contains the translation table of the IPU's L2-MMU (compare Figure 4.16).

In Figure 4.17, the attack sequence is depicted. In the following the subsequent steps in the sequence is described.:

- [1]: Craft and write firmware to firmware load address (FLA) $addr_{FLA}$
- [2]: Gain access to memory partition *Part2* of victim's memory space. The translation table is allocated and initialized by setting each entry to the *PA*

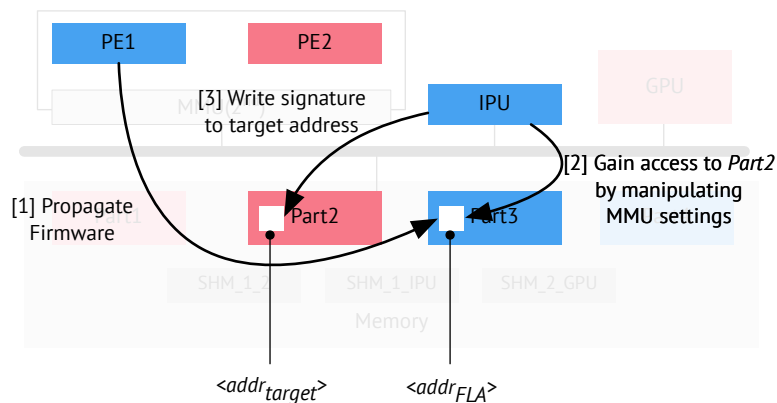


Fig. 4.17 Exploitation firmware attack sequence.

where the firmware is located ($addr_{FLA}$). Next, the PA $addr_{target}$ of the victim-domain's memory-partition is written to the translation tables last entry (or any other available)

[3]: The exploit code writes the signature $0xEEAADDCC$ to $addr_{target}$

Temporal Vulnerability Score

The temporal vulnerability score considered in this section refers to the base score given in Table 4.8. Accordingly, the corresponding Base Score is 8.1. As a result, the penetration test is rated to 7.4, which is still rated as high severity. As is shown in Table 4.9, the temporal vulnerability score manipulates in a positive direction and limits the severity.

Table 4.9 CVSS temporal score of the co-processor misuse pentest.

Temporal Metric	Rating	Multiplier
Exploit code maturity	Proof-of-Concept	0.94
Remediation level	Unavailable	1
Report confidence	Reasonable	0.96
Temporal Score	High	7.4

Rationale The rationale to the scoring of the exploit code maturity is fairly similar to the previous penetration test given in 4.1.2.

Exploit Code Maturity In this particular case, either the score *Functional* or *Proof-of-Concept* is applicable. Other score values, such as *High* do not apply, for example, since there is no autonomous code made available. However, since the pentest has shown, it is feasible to mount an attack and exploit co-processors on this particular system level. Accordingly, the score value *Proof-of-Concept* is applicable. During the penetration test, source code has been produced which is capable of testing a hypothesis⁸. This code has not been made public. However, the technique and the conceptual consideration is practical to similar systems. Since the exploit code is not runnable with the interaction and manual work of the penetration tester, it is unreasonable to apply the metric value *Functional*.

Remediation Level The metric value *Unavailable* is reasoned by the fact that no solution is applicable at the intermediate system level. The remediation is subject to the proposed countermeasure that is part of the following chapter. However, DMA based access control issues are encountered in many approaches applying, for example, hypervisors and para-virtualized drivers. Therefore, a workaround which involves one of those solutions might be applicable. In such cases, the remediation level could be set to *Workaround* and the overall temporal score is multiplied by 0.97 instead of 1.

Report Confidence To the date of discovering the vulnerability, no evidence indicated such impacts to the integrity of AMP-based systems. However, the root-cause of exploit technique to be effective based on well-known issues in DMA attack scenarios. Therefore, the metric value is set to *Reasonable* rather than *Unknown*.

4.3 Generalized Vulnerability and Exploitation Pattern

In this section, the previously analysed vulnerabilities and attack vectors are generalized and defined as a common applicable pattern. The pattern analysed here takes the design paradigms of AMP into account and examines the differences to well-known SMP based approaches.

⁸Compare Section 4.2.2

4.3.1 Modelling Protection Architectures

In the following a model is proposed which describes the protection architecture in multi-layered system architectures. The system is horizontally divided into a set of logical layers each of which contains a certain number of entities. Vertically, each entity belongs to a trust boundary. In the following, the elements of the architecture are described in detail.

Entities. The entities are distinguishable into three categories. These are subjects, objects and TSF. Subjects represent active entities such as processes, OSs, or hardware elements, depending on the respective layer. Subjects will be denoted by \bullet . Objects (denoted by \blacksquare) are passive entities which are either shared or exclusive by subjects. TSFs instantiate protection mechanisms such as access control, denoted by \blacktriangle .

Layers. A system consists of l layers forming a privilege hierarchy. The hierarchy is defined that the lowest layer comprises the highest privileges. Layers are denoted as the subscript of an entity. For example: \bullet_l . The layers are in conjunction with the layered architecture introduced in Section 2.1.2.

Trust Boundaries. Trust Boundary (TB) are in analogy to AD and describe a logical assignment to a particular group of entities. TBs are denoted as the superscript on an entity. For example: \bullet^{tb} .

Relationships. Entities interact with each other. In this case, the accesses, forming a read or write for example, are denoted by \rightarrow . For example: $\bullet \rightarrow \blacksquare$.

Graphical Visualization. In Figure 4.18, the concept is visualized. TBs are visualized by the colouring of the entities. The example shows three TB: red, green and blue. According to the defined denotation, here the subjects \bullet_{l+2}^{red} and \bullet_{l+2}^{blue} , the objects \blacksquare_{l+1}^{red} and $\blacksquare_{l+1}^{blue}$ and the TSF \blacktriangle_l^{green} are visualized. In the visualization the access relationship $\bullet_{l+2}^{red} \rightarrow \blacksquare_{l+1}^{red}$ and $\bullet_{l+2}^{red} \rightarrow \blacksquare_{l+1}^{blue}$ are shown.

4.3.2 Preliminaries of an AMP system

In this section the preliminaries for defining threat detection rules.

Subjects: $\bullet > 1$

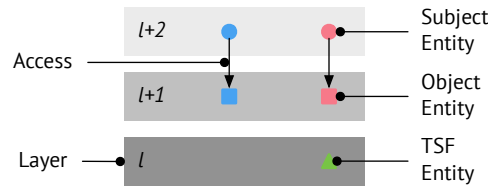


Fig. 4.18 System layers and entity relationships.

Objects: $n > 1$

Layers: $l > 1$

Trust Boundaries: $tb > 1$

4.3.3 Breach Access Controls on Intermediated Level

Security Problem Statement

The issue of breached access control mechanisms is not new in certain areas. As was mentioned before, in traditional hypervisor-based SMP-systems this issue, particularly the DMA problem, is discussed thoroughly.

All proposed memory access breaches are based on the same concept. They exploit a missing access control mechanism in the co-processor implementation. On the master PE side, the access control is implemented by the second-stage MMU, which is commonly part of the virtualization extension of modern processors. However, a problem arises if PEs are implemented which do not thoroughly support the concept of AMP, and accordingly, properly managed access control on the intermediate layer.

In general, the root-cause of attacks in a memory-mapped system is the UMA architecture. Regardless of the means to separate the shared resource, it is still a single element which is operated by multiple AMP-domains. The aim of the memory-map AMP approach is to virtually facilitate a NUMA architecture. In the same fashion, UMA provides functional advantages on the contrary. Amongst others, the closed-coupled application can share data very efficiently. Nevertheless, most MPSoCs implement an UMA memory architecture, and therefore those are the focus of the attack analysis.

In an access control scenario, multiple subjects request access to objects. The subjects are instantiated entities. Accordingly, the objects are resource entities. Fur-

thermore, there must be an enforcement entity which applies a policy. In a layered system, each of the entities resides on a particular level.

Concerning the given problem of direct memory access of co-processors, the system looks like as shown in Figure 4.19. Two entities on the higher layers ($ln + x$) access the resource entity on the lowest level (ln). One of the entities is, like in the analysed exploit, the master PE and a co-PE, which are denoted by *RequestingSubject1* and *RequestingSubject2* respectively. The difference between both accesses is the policy enforcement entity (green triangle) on the intermediate layer denoted by $l + 1$. Whereas *RequestingSubject2* does not apply a policy enforcement (such as a MMU), *RequestingSubject1* is intercepted by such.

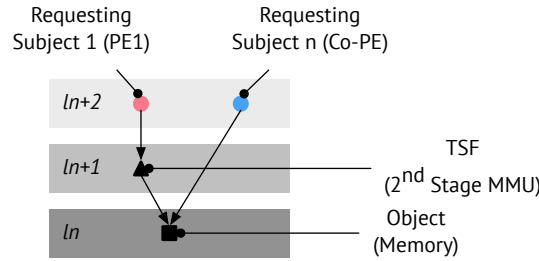


Fig. 4.19 Generalized access control problem.

Although the general access control problem would lead to serious attack surfaces, this issue might have arisen from a changed assumption on the system design. Formerly, the system architectures incorporated a traditional hypervisor which utilised the hardware synchronously. Every request or operation would be interceptable by the hypervisor. That means the hypervisor has been assumed to be the single point of policy enforcement. As an example, concerning the GPU exploit scenario mentioned earlier, the hypervisor could have been empowered to validate or enforce the in and output descriptors to prevent a manipulation of the higher system layers. Since this instance does not exist, or should not exist, in the targeted AMP environment, the enforcement of the access policies must be performed in a different place.

Decomposition

Here, a rule-set is introduced to detect the security problem. The rules apply only to the preliminaries given in Section 4.3.2.

EoP_1: Access from adjacent trust boundary. An EoP threat occurs, when a subject that belongs to an different trust boundary as the object. In this case, the layer is considered to be the same.

$$(\bullet_l^n \rightarrow \blacksquare_l^n) \wedge (\bullet_l^m \rightarrow \blacksquare_l^n)$$

EoP_2: Access from higher layers. An EoP threat occurs, when a subject that belongs to layer that is higher than the object's layer.

$$\bullet_{l+1}^n \rightarrow \blacksquare_l^n$$

EoP_3: TSF on a different layer than the object. An EoP threat occurs, when a TSF that belongs to layer that is higher than the object's layer. Otherwise, a similar situation as shown in *EoP_2* occurs.

$$\blacktriangle_{l+1}^n \rightarrow \blacksquare_l^n$$

EoP_4: Subject uses TSF that belongs to a different trust boundary.

$$\bullet_{l+1}^n \rightarrow (\blacktriangle_l^m \rightarrow \blacksquare_l^m)$$

4.3.4 Denial-of-Service a Shared Resource

Security Problem

Here, availability aspects of objects are considered to be threatened. If two or more subjects (entities) utilize the same resource negative effects could occur. According to a threat identification using a DFD, this can be identified as shown in Figure 4.20.

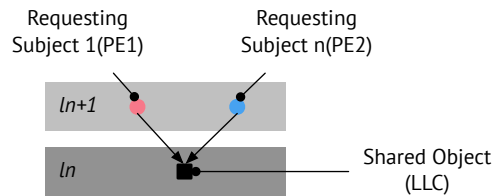


Fig. 4.20 Resource sharing.

According to the shown figure, the problem arises when two subjects out of two distinct trust boundaries access a resource. It is assumed that the subjects and objects reside on the same architectural layer.

DoS_1: Two subjects of different TB share a resource without applying a TSF.

$$(\bullet_{lx}^n \rightarrow \blacksquare_{lx}^p) \wedge (\bullet_{lx}^m \rightarrow \blacksquare_{lx}^p)$$

4.3.5 Identification of Attack Surfaces in AMP Systems

The assessment throughout this chapter has shown, the hypothesised attacks are conceptually and practically existent. There are two branches; one might think of, to conclude and proceed on the foundation of this contributions. Either, the particular problems will be solved in dedicated solutions, or the results will be turned into rules to prevent new architectures from being prone to such security problems.

The former takes the details of the results into account when it comes to the conceptualization of solutions to the revealed security problems. Here, the same assumptions and architectures build the reference for such solutions⁹. The latter seeks for design rules, patterns, tactics or formal models.

Security Models Aiming for Isolation

In the following security, models are elaborated aiming for a strict separation of architectures in order to prevent security problems assessed in this work.

The main purpose of introducing layers and entities is motivated by the *separation of protection and security* principle [187]. The principle seeks to distinct an actual protection mechanism from security policies¹⁰. With this intention, here it is defined what the general directives (policies) for a layer are. The properties are hence derived from security properties such as confidentiality, integrity, availability, accountability and authenticity. Consequently, the layer characteristics reflect those corresponding security properties. Defining characteristics for system models is not new to certain areas. Particularly in the field of AMP systems security models such as Multiple Independent Levels of Security (MILS) have been introduced to model the security of high assurance systems especially for aircraft, space, and defense purposes [5, 180]. MILS focuses on properties including data isolation, control of information flow, periods processing and fault isolation [182].

⁹This is how it is done in the risk treatment chapter 5

¹⁰Policy is treated as a directive.

Data isolation: The isolation of data does not only affect the confidentiality of information, it is also aimed at the separation capabilities of the system design.

Control of information flow: The location of the information has to be defined. Furthermore, access control has to be enforced.

Periods processing: This aims for single core or classical virtualized systems which share one or more processors. In this case, it has to be proven that each system can meet its timing requirements.

Fault isolation: Failures are detected, contained and recovered locally [182].

These properties relate to the definition of separation kernels [128, 140]. The intention of kernels is to define properties for TSF in order to achieve a particular degree of assurance. These properties include [128]:

- Protection of all resources (including CPU, memory and devices) from unauthorized access
- Separation of internal resources used by the TSF from exported resources made available to subjects
- Partitioning and isolation of exported resources
- Mediation of information flows between partitions and between exported resources

Each entity of a layer must be delimit-able to other entities. This is applicable to for two dimensions. Horizontally, it shall have defined boundaries to entities at the same layer. Orthogonally, the vertical separation contributes to a defined boundary and therefore interface to other layers. This property contributes to the security properties integrity and confidentiality. In order to make an layer entity tangible in their environment it needs to be identifiable. Identifiability is a key property to organize and administer large systems. Furthermore, with respect to security this property is essential when it comes to authentic communication. Availability is from many perspectives an important quality and therefore a crucial property of a system layer. Availability means that the resource consumption is well defined and controllable. When it comes to resources, particularly shared resources, the privileges to operate on them are considerable. It must be distinguishable if an requesting entity gets access to a resource granted which is either horizontal or vertical. This property is quite complex since it implies the combination of the above mentioned properties. An identification of an requesting entity is needed as well as the delimitation of the resource itself.

For the facilitated security policies, the protection mechanisms to following attributes must be applicable. Protection mechanisms shall be:

- non-by-passable
- evaluable
- always invoked
- tamper proof

These are commonly known as the acronym *NEAT* [4].

In the EURO MILS CC PP [50], the authors require particular hardware facilities in order to realize the TSF. These facilities include IPMMUs, MMUs, etc. That means, the profiled ToE in this case, aims for a software-based solution, being hardware agnostic as much as possible. However, the assessment in this chapter has revealed that the security problems are due to architectural design flaws in the hardware which is due to the AMP (Type-0 Hypervisor) construction, rather than implementing a hypervisor and SMP. As a conclusion, if the separation capabilities are moved to the hardware, the hardware, in turn, has to realize all necessary functions with the properties as they are introduced in this section. Meaning, the hardware is the separation kernel then.

4.3.6 Primary and Secondary Assets

In the first place, this work focuses on *tampering* threats on the memory partitions of the MC-domains. Concerning the example as given in Figure 5.2, one MC-domain tampers with the memory partition ($Part_i$) of another domain. Despite, the compromising of the integrity of the main memory, this can be the root-cause for further or even more advanced threats. It is worth mentioning that tampering might lead to elevation of privileges or one subsequent step of spoofing a communication link to another entity, even though, denial-of-service attacks can be mounted by tampering with the memory-base of a system. This is motivated by the fact that software intensive systems rely on their code and database stored in the main memory. Tampering with that does not only compromise and modify information, but also the control flow integrity of their function. This means, that by having the ability to change the control flow deliberately. For example, an adversary might gain full or partial control of the vehicle's behaviour. Impacts on safety and operation of the entire vehicle are severe. As a result, memory storage is a valuable asset. In the following section, the potential attacks on memory assets are structured.

4.3.7 Attack Objectives and Scenarios

This section describes the AMP system-specific technical attack objectives. As is discussed in Section 4.3.7, threat actors aim for changing the system state. In other words, the adversary wants to control or disrupt the system's behaviour. Disregarding of their motivation, it is assumed that the adversary aims for a specific function. A specific function which is technically represented as a process on the application layer and logically bound to a specific MC-domain (compare Section 4.3.3).

Harper et al. [62] define five phases of hacking/attacking a systems, from which the first three are considered in the following: *Reconnaissance*, which means to "obtain information either actively or passively" [62, p. 15], *Scanning*, to identify the environment which will be compromised, *Gaining Access*, which means to exploit the previously identified vulnerabilities [62], and *Maintaining Access*, or to make sure a re-entry is possible and finally to *cover tracks* to hide the malicious activities.

A crucial fact of considering memory based (and memory-map based) attack vectors is that the attacker needs to know its particular target memory area and therefore its address. Many known memory exploits and techniques rely on this issue. Some prominent examples are buffer overflow attacks [40], format-string attacks [40] and Return Oriented Programming (ROP) [24]. These techniques have one common base, the modification of a return address of a function call in order to jump into injected and malformed code sections. The details of those techniques are out of the scope of this work. However, it is important to mention that the effort to determine the specific address or the relative position (offset) to a particular instruction is crucial. Usually the attack vector works as follows. Somehow the adversary needs to gain access to the application and exploits a vulnerability of a function, via a buffer overflow for example. In the next step, the adversary re-interprets the code structure and seeks for a suitable position to interfere with the control flow of the application. Most commonly this is done via overwriting the return address of a function call. The newly written address points to a memory area which is under control of the adversary.

In the following this process is formalized.

Step 1: Gain access/entry the function

Step 2: Re-interpret structure

Step 3: Mount attack

In order to mount an attack described above, there were still some assumptions made. In order to find a suitable entry to the system, the adversary needs to put effort

into exploring it. This is usually referenced to as reconnaissance in a preparational step. Reconnaissance also takes place in Step 2, when the adversary seeks to re-interpret the structure of the system to proceed with Step 3, which is the execution of the attack. As a result, this could be split into preparational reconnaissance and online reconnaissance.

Threatening the System State. This subsection aims at defining the goal of threatening a system element. In this particular environment, the fulfilment of system qualities is prioritized. One important component in achieving this goal is to ensure that the system operates as it was intended. Hence, the proper function of the system turns into the focus of the stakeholder of the system. On the threat agent side this means that if someone endeavours to misuse the system in any way, they must manipulate its behaviour. The system state is, on a logical level, the asset that an adversary aims at changing in order to reach their goal. The system's state is an abstract description of what a system or function is operating on during runtime. Consequently, if a threat agent is capable of changing the state of a system, he has control over the system. If this is transferred to AMP systems, OS have states which are exposed to be fraudulently interfered with by other OS or data flows consumed by them.

Control-Flow Integrity (CFI) related Attacks

Attacks on the integrity of control-flows are a severe representation of tampering threats. This type of attack is affecting the Control Flow Integrity (CFI) by subverting the machine-code of software-systems [2]. In short, CFI attacks aim for illicit control of a program's state. Control-flows can be subverted either directly by tampering with the instructions of the machine code or indirectly by modifications of data that are consumed by the program. The latter type of attack is referred to as non-control flow data. Control-flow data are kernel text sections, such as instructions in binary code or function pointers which redirect the execution flow of instructions. Furthermore, processor registers which contain the instructions to be executed [133]. Non-control flow data preserve possibilities to interfere with the state of the program and can be categorised with the following [26]. Typical examples concerning MPSoCs are input data which are consumed by a program and affects their behaviour. Input data can be something that has been issued by an external entity such as a sensor connected by an automotive bus. On an application and OS level, this might be a message passed over an SHM or on a SoC level data frame that was communicated through the communication architecture. The subversion of configuration data includes a broad

range of possible surfaces as well. Most notably, this could be the configuration of a processing element or a memory protection unit which enforces the separated address spaces for the particular system levels. Modifying identity data implements typical spoofing threats. The modification can cause a program to falsely authenticate a communication entity due to compromised identification data. Lastly, the category of decision-making data can influence boolean variables which imply a conjunction or disjunction to reach the final verdict [26].

CFI Attack Types

Following Petroni et. al [132], the following CFI-specific Threats are categories, which are differentiated and considered:

User-space object hiding: Aims to inject code with an illicit capability within an user-space object such as an application.

Privilege escalation: Code injection with illicit capability within user-space aiming for escalating/elevating privileges.

Re-entry/Backdoor: Implements illicit capability on OS-level.

Reconnaissance: Reverse engineering of kernel structures in order to gain knowledge about system structures or functionalities.

Defense neutralization: Deactivation of defense or protection controls implemented on OS-level.

Control Flow Side Channel Attacks In general, side channel attacks aim at exploiting leaked information which was gained by a covert channel [124]. Covert channels are not intended for information transfers such as the software program's effect on the system load [105]. Recent endeavours revealed several instances of side channel attacks that take effect on adjacent system entities. For example, *Rowhammer* has been introduced by Kim et al. in order to flip bits in a DRAM¹¹ row without accessing them directly [96]. Furthermore, with *Meltdown* and *Spectre*, possibilities have been demonstrated as how to misuse the speculative behaviour of modern (embedded) processors [103, 111].

¹¹Dynamic Random Access Memory

4.4 Summary

This chapter elaborated upon the vulnerability assessment of two hypothesised attacks. This was approached by the conceptual analysis of the vulnerability and exploitation of these attacks. First, the disruption of the accesses to a shared last level cache was examined. Second, the assessment on the tampering of memory by an adjacent processing element has been conducted.

A denial-of-service attack on shared cache setups of modern processing element designs has been discussed. The contention is caused by shared way-sets in k-way-set associative caches. The concept shows how to provoke the contention by overcommitting associated way-sets. As a result, the vulnerability is a non-controlled shared usage of cache way-sets, which enables an adversary to provoke cache-misses on memory accesses. In the CVSS scoring, this vulnerability has been scored at *Medium* (5.3) with a high impact on the availability of the attacked domain.

The exploitation of the identified vulnerability has been conceptually and experimentally approached. The assessment reveals that particular physical addresses can be modified and cause the previously described effects. This is possible, even though they are in separated memory partitions of asynchronous domains located in an AMP system. In a penetration test, the formerly conceptualized attack was experimentally demonstrated. Here, on the basis of the experimental platform, an increase of *129,501 CPU cycles* (arith. mean) per memory access has been provoked by the DoS attack. The consideration of the temporal CVSS score slightly decreased the value from *5.3* to *4.8*, which is still a *Medium* exploitability score. The maturity of the given experimental exploit code has been rated as *Functional*, which means it shows the effects but must be adapted to be applicable in other systems.

The second assessment approached concepts of misusing a memory access capability in heterogeneous MPSoC designs. DMA capable peripherals such as GPUs and co-processors can be misused to exploit an insufficient protection architecture within the MPSoC. The conceptual exploitation has been analysed and scored with CVSS at the level *High* (8.1). Having this vulnerability in place, a change of privilege scope is possible. This leads to *High* impacts regarding the confidentiality and integrity of the asset.

Experimentally, the exploitation of the identified vulnerability has been explored and scored. The penetration test demonstrates a compromise of a co-processor firmware image to circumvent the memory protection of the main processors. As a result, the

temporal CVSS score is *High* but decreased to 7.4 due to the maturity of the shown exploit code.

Both assessments were reflected in the third part of this chapter. Here, a general view of the given vulnerabilities and assets is conducted. For that purpose, the concepts have been generalized and reflected in the layer hierarchy of an AMP system. The results are twofold. First, the systems should be analysed to reflect the distinct hierarchies of an AMP system. This will reveal resource control and access violations. Furthermore, the findings enable the ability to define requirements for protection architectures such as multi-layered systems.

Risk Treatment

Exploitation Prevention and Mitigation Concepts

Contents

5.1	Risk Treatment Strategy	146
5.1.1	Target and Residual Attack Potential	147
5.1.2	Exploitation Prevention on the Intermediate Layer	147
5.1.3	Security Solution	148
5.1.4	Primary and Secondary Countermeasures	152
5.2	Countermeasure 1: Memory Domain-Blocks	154
5.2.1	Effectiveness Requirements	154
5.2.2	Mitigation Concept Analysis	154
5.2.3	Proof-of-Concept Implementation	157
5.3	Countermeasure 2: Memory-Map Shuffling	160
5.3.1	Effectiveness Requirements	161
5.3.2	Mitigation Concept Analysis	161
5.3.3	Case Study	168
5.4	Summary	170

"We can't solve our problems by using the same kind of thinking we used to create them"

Albert Einstein

Reacting to observed vulnerabilities and novel attack vectors is a widely discussed field in the security engineering discipline. Usually, new observations trigger the engineers to come up with new countermeasures or architectural designs. This cat and mice game is constantly changing the advantage between the contestants. Particularly, the stakeholders of a system have aimed at keeping the advantage or the upper hand for the adversaries. In general, this chapter considers the following research question:

Research Question 2: *How can one mitigate the risk of exploitation of the previously identified vulnerabilities?*

The research question is approached by connecting to the generalised vulnerability analysis given in the previous chapter (compare with Section 4.3). The bottom-up/inductive approach is in conjunction with the identified DMA and cache vulnerabilities. A proposal to mitigate these specific weaknesses is conceptually analysed. The chapter begins with an architectural proposal which enables future designs to be defined from the top-down/deductive perspective.

After an analysis of the interdependencies between the vulnerability and its countermeasures, first, a definition of protection categories in layered architectures is given. This is considered as the foundation for the proposed mitigation approaches and aims for the context of exploitation prevention on the intermediate layer. This approach aims for transferability and are suitably abstracted to apply to other appliances. The observed mitigation patterns will be generalised with a protection architecture model, which seeks to abstract the key findings and empower future architectural designs.

The contribution in mitigating cache-thrashing is published in Schnarz et al. [144, 146]. Furthermore, the concept of memory-map shuffling is proposed by Schnarz et al. in [147].

5.1 Risk Treatment Strategy

On the basis of the hypothesised attacks, here, the risk treatment is discussed. This includes the treatment strategy as well as functional and assurance requirements.

According to the risk levels that result from the risk assessment, the general strategy to treat the risks is *mitigation*. As a result, the mitigation shall be facilitated as a

countermeasure that functionally contributes to the reduction of the overall risk. This shall be measured and therefore assured by appropriate techniques.

Other risk treatment strategies are not considered at this level of abstraction. This is motivated by the nature of the other risk treatment strategies. Risk avoidance, for example, would imply a degradation of the functionality (applications) running on the ToE. However, if the proposed mitigations are not capable of reducing the risk properly, the result might be a limitation of the severity of functions that are executed by the ToE. Furthermore, considering risk transfer, organizational means for handling the risk are out of scope. Risk acceptance will play a role in the statement of the residual risk as part of the result in the evaluation of the mitigation approaches.

5.1.1 Target and Residual Attack Potential

The targeted AP is defined concerning the definition given in Table 2.5. According to the risk level of 4, the required resistance to the attacks shall be greater or equal to *Enhanced Basic* (AP range: 10 – 13). The residual AP should be greater or equal to *Moderate* (AP range: 14 – 19).

5.1.2 Exploitation Prevention on the Intermediate Layer

How to prevent from the exploitation of a vulnerability depends on the particular stage in the product lifecycle. For example, if the product is in the design and development phase, an observed vulnerability can be fixed by design changes or the implementation of explicit countermeasures. However, if the product is already deployed, the fix of vulnerabilities implies much more constraints. This section aims at analysing and defining particular protection strategies in order to describe the dependencies between vulnerabilities and the different ways to prevent them from exploitation.

After a vulnerability in the architecture has been observed, exploitation prevention mechanisms are the last line of defence to hinder an adversary from compromising the aimed for asset successfully. Exploitation prevention mechanisms are countermeasures which are facilitated due to expected but unknown vulnerabilities. These mechanisms should apply when another protection mechanism has failed. That means exploitation prevention measures are in place to increase the hurdle (effort) of implementing an exploitation in order to penetrate security mechanisms.

As a result, there are primary and secondary countermeasures. The former aims at controlling and protecting a particular asset explicitly. Primary countermeasures are realized due to known and hypothesised/anticipated threats and attacks based on

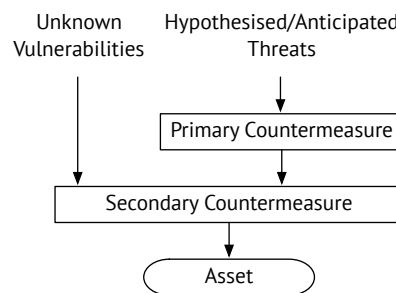


Fig. 5.1 Primary and secondary countermeasures in relation to exploitation and asset.

the assumptions made during the design of the system. However, the latter aims at countering an intrusion in case the primary measure has failed, or a new vulnerability will be exploited. In Figure 5.1, this principle is depicted.

Defence-in-Depth. In complex systems consisting of several layers, usually the security architecture facilitates a defence-in-depth philosophy.

"A defence in depth design philosophy proposes multiple levels of security devices (...). The greater the value of the information assets within a given risk domain, the greater the number of layers of security technology must be penetrated to reach those information assets." [14, p. 518]

Bidgonli defined the defence-in-depth philosophy for the appliance of large enterprise environments, which connects a wide set of Commercial Of The Shelf (COTS) devices. Concerning the multi-layered system architecture, the defence-in-depth philosophy is applicable from two perspectives. Each layer can facilitate its individual protection mechanisms. The stack of architectural layers and the applied protection mechanisms represent the *depth* of the defensive architecture. Furthermore, the previously mentioned differentiation between countermeasures applies as well. Each of the facilitated protection mechanisms facilitates either the concept of primary or secondary countermeasures.

5.1.3 Security Solution

This section aims at describing the dependency between a vulnerability and a countermeasure. The dependency will close the gap between the security problem and its solution. Therefore the security pattern will be completed by the countermeasures proposed in this chapter. The context of the of the security pattern is defined in the

ToE description of this work (compare with Sections 2.1 and 3). Additionally, the particular security problems were examined in Chapter 4. Here, the considered threats and attacks to the given context are stated. In addition to that, the security problems were quantified by applying the concept of AP and the CVSS.

The only missing point is the selection of a suitable protection mechanism, which is the security solution. Many ontologies exist that map security problems or objectives to security solutions. A prominent and well-accepted collection [149] of those mappings is stated in the CC SFR. It is recommended to choose from the SFR families and components functions that satisfy the security objectives [43]. The components include:

- Logging and audit
- Identification and authentication
- Cryptographic operation
- Access control
- Information flow control
- Management functions
- Protection of user data
- Protection of TSF
- Protection of (user) data during communication with external entities

However, there is no formal guidance on how to choose suitable functional requirements for a particular problem. This relies only on the author and evaluator of ST or PP. Furthermore, the definition of the STRIDE model also states at high-level categories to mitigate identified threats. Here, tactics are identified to approach the particular threats. This is shown in Table 5.1.

Table 5.1 Tactics for threat mitigation (summary of [154, p. 145ff]).

Threat	Property lated	Vio- Countermeasure
Spoofing	Authenticity	Authentication by cryptographic means (message authentication codes).
Tampering	Integrity	Implementing permission system or cryptographic means.
Repudiation	Non-Repudiation	Digital signature systems.
Information Disclosure	Confidentiality	Implementing permission system or cryptographic means (encryption).
Denial-of-Service	Availability	Implement resource limitation or redundancy.
Elevation of Privilege	Authorization	Implementing permission system.

The previously given examples for choosing suitable functional security requirements will be taken into account for the countermeasures proposed in this chapter. However, since the security problem of AMP systems are specific to its context, security patterns for the ToE are stated in the following sections. These protection architecture tactics are the result of the key findings of the particular countermeasures.

Security Solution Approach for a Breached Access Control

The solution to the security problem introduced in Section 4.3.3, is elaborated upon in the following. The here given solution shall enable an architectural design which will not lead to the identified threats. This is in contrast to the proposal given in Section 5.3. There a solution is shown that applies to systems that cannot be redesigned.

In Figure 5.2, the hardened solution is depicted. In the following, paragraphs the design rules are elaborated and referred to the threats.

A TSF shall reside on a lower layer than the requesting subject. In order to trust the TSF, it must be placed at a layer that is more privileged as the subject, requiring access to an object. This shall ensure the integrity of the security solution (*EoP_2* and *EoP_3*).

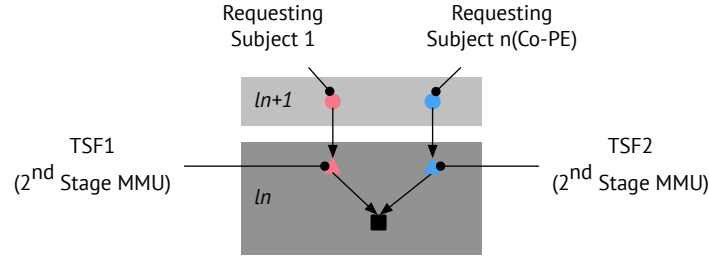


Fig. 5.2 Distributed access control.

For each TB there shall be a TSF for a single object. In order to prevent the circumvention of a TSF by subjects of adjacent trust boundaries, each TB shall implement a TSF. This prevents EoP by a untrusted subject in a adjacent TB (*EoP_1* and *EoP_4*).

TSF and Object shall reside on the same layer. The requested object shall be controlled by a TSF that resides on the same layer. This prevents the EoP across privilege layers (*EoP_2*).

Example of an optimized technical architecture. The resulting technical architecture must include a mean to protect the memory from requests issued by PEs that have an interface to the system memory. In Figure 5.3, the exemplary architecture is enhanced with a MPU for each PE. Those MPUs are facilitated on the intermediate layer. Every violent access request would be trapped by the respective MPU. This concept of a distributed access control architecture would enable a throughout AMP domain design. No protection mechanism would rely on a higher level system element.

As a result of the given problem analysis, a properly crafted MPSoC design the access control mechanisms shall be distributed as it is shown in Figure 5.2.

Security Solution Approach for DoS of a Shared Objects

Not in all cases, a total separation of objects is possible. This is particularly true for LLCs, so it has been analyzed and demonstrated in Section 4.1. In general, a suitable approach to mitigating a DoS attack to a resource is the arbitration of accesses. In other words, a TSF shall handle those accesses and enforce a balanced (or controllable) scheduling between competing subjects. The rules to instantiate such an TSF is discussed in the section on access control (compare Section 5.1.3). From a functional

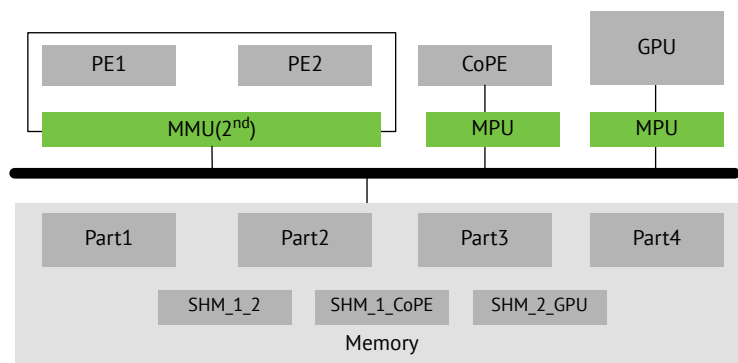


Fig. 5.3 Modified MPSoC architecture to prevent from direct access misuse.

point of view, here resource arbitration or resource scheduling could suitably to be applied [98, 99].

However, particularly for caches, this is not always possible without changes in the design of a cache. Therefore, here the decomposition approach is proposed. This means, that the object will be decomposed in its sub-components. On this, more detailed, representation, possibly sub-objects exist that can individually be handled (compare Figure 5.4). This general approach of de-compositing is utilized to present the domain-block concept in Section 5.2.

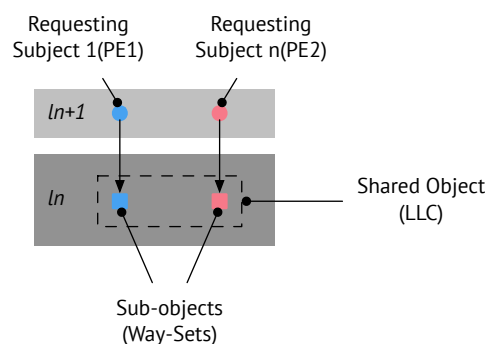


Fig. 5.4 Object sharing security problem model.

5.1.4 Primary and Secondary Countermeasures

According to the memory-map based AMP construction, the MMU capabilities are utilisable to facilitate the concept of primary and secondary countermeasures. The memory-map is comprised of a central point in the configuration of the hardware.

Memory-Maps as Primary Countermeasure. The intrinsic concept of memory maps is clearly stated. It shall organize the physical memory by utilizing distinct address spaces¹. From the security point of view, the memory protection capabilities of a MMU turn into focus. The memory protection facilitates the concept of access control to a resource. In Figure 5.5, the principle is visualized. The example is based on the driver information case study and applies two asynchronous domains which access the common memory resource. Accordingly, the *PEs* are the subjects that request access the object, which is the memory. The manifest which keeps the security policy is represented by the memory-map table. The actual enforcement for this policy is the MMU device.

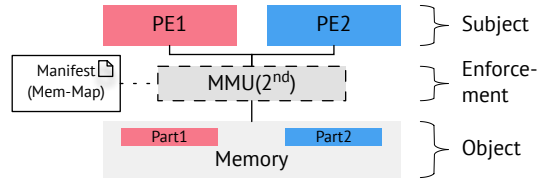


Fig. 5.5 MMU utilized as means of access control.

Memory-Maps as a Secondary Countermeasure. As a result of the previous analysis, the capability of secondary countermeasures was not currently considered on the intermediate level of AMP-based systems. Hence, this work proposes the integration of secondary countermeasure into the intermediate level (compare with Figure 5.6).

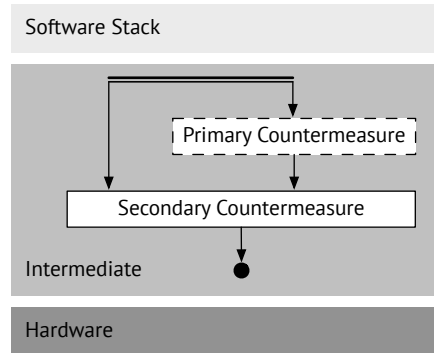


Fig. 5.6 Integration of a secondary countermeasure on the intermediate level.

Amongst the primary access control enforcement of the MMU, here the address translation capability is utilized. In Section 5.3, there is a concept to re-organize the

¹compare Section 2.1.4 for a detailed explanation

memory-map between the IPA and PA address mapping. Thus, the localization of targeted physical memory pages shall be mitigated without knowledge of the mapping table.

5.2 Countermeasure 1: Memory Domain-Blocks

The general mitigation approach aims at separating the WSs in the LLC in order to avoid the impact of deliberately caused competing accesses. Therefore, the approach shown here, adopts the concept of cache colouring and applies it to the concept of intermediate layer memory-maps in an AMP-based environment. This results in a technique for creating the second stage (IPA to PA) mapping.

Cache colouring is a technique that exploits the cache associativity to avoid cache contention issues. Memory pages are tagged (coloured) according to their association with the cache-lines in the LLC. The entity in the system which allocates those pages to processes seeks to assign distinct processes with different coloured pages. Since this allocation technique is mostly applied on the OS layer, this is often part of the memory allocator implementation of OS. For example, in Linux operating systems, the *SLAB* allocator calculates offsets for optimised cache contention [56].

In the following, a domain-block mapping is introduced to direct the access of distinct ADs or MCs to a subset of the WS in the cache. By this token, a logical separation of the shared LLC is achieved.

5.2.1 Effectiveness Requirements

Conceptually, the method proposed here should fulfil and answer research question *RQ2* while also concerning the general aim of this research. The general aim requires finding solutions adapting the intrinsic configuration interface of an AMP system, which is the memory map. The countermeasure should implement the concept of a primary countermeasure.

The proposed solution shall mitigate the risk of denial-of-service of a shared LLC. The proposed solution shall avoid the $t_{penalty}$ with regards to memory accesses. According to the case study the targeted AP is Moderate.

5.2.2 Mitigation Concept Analysis

Domain-blocks represent a consecutive chunk of memory which belongs to a particular AD. This memory chunk is divided into MLs each of which correspond to a particular

WS in cache². The index of the ML (ML_i) directly indicates the index of the WS (WS_i) and is calculated by:

$$ML_i \bmod v = WS_i$$

The domain-blocks can be sized and properly placed in the main memory so that the domains do not share any WS in the LLC. This concept is visualized in Figure 5.7. The figure shows a domain-block mapping applied for two domains (MC1 and MC2 concerning the case study). According to the concept of page colouring, there are two distinct colours used in this example. In the physical memory, an alternating pattern of domain-blocks is arranged. The blocks coloured in *blue* correspond to domain *MC2* and the red blocks to domain *MC1*, respectively. The size of each domain-block defines how many WS are associated with a domain. In the given example, each domain shall have access to half of the LLC. Thus, the number of WS (v) is divided. Accordingly, in the example, each domain-block consists of $v/2$ ML. Within a particular domain-block the first ML (index ML_0) is associated with WS_0 in the LLC and the last $ML_{(v/2-1)}$ with $WS_{(v/2-1)}$. The number of ML per domain-block cannot be chosen freely. There are dependencies to the granularity of page-sizes which can be addressed by the MMU. This aspect is discussed in the following sections.

Sizing Domain-Blocks. In the following, the parameters are discussed to size and arrange domain-blocks in general. Table 5.2 shows all key parameters for defining domain blocks. It, therefore, enhances the terminology given in Table 4.1 within the attack consideration.

As it is mentioned previously, to implement the domain-block concept some dependencies to the memory page granularity have to be considered. Memory pages are the mappable memory chunks a MMU usually handles. Size (denoted by P_{Size}) and granularity of such pages are architecture dependent. For example, a common page size granularity is 4KiB [150]. In general, this influences the concept in two ways: first, the smallest size of a domain-block equals the number of ML a memory page contains. This number of ML also defines the number of WS in the LLC that belong to a particular domain. Accordingly, the number of consecutive WS in the LLC that are assigned to a particular domain is:

$$l = \frac{P_{Size}}{ML_{Size}}$$

²Refer to Section 4.1.1 for caching terminology and design.

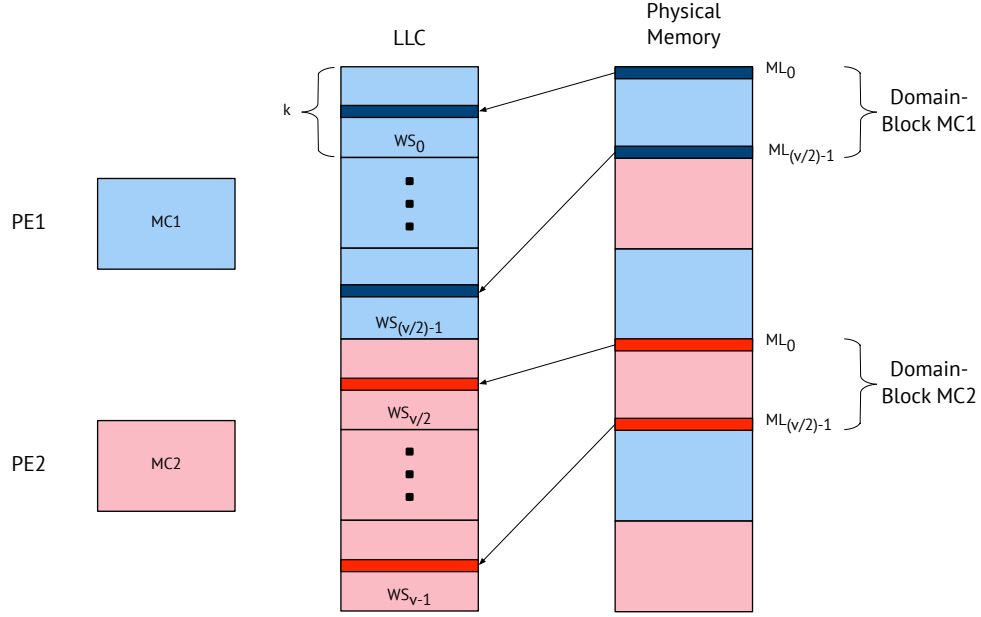


Fig. 5.7 Principle of domain blocks with exemplary mapping of main memory.

The second influencing aspect for this concept is: since the domain-block concept relies on the separation of WSs, the number of supported domains depends on the WS number in the LLC. In the example given above, only two domains compete for the shared cache. In the following, the general limits are given. The maximum number of WSs per domains (w_{max}) supported by the concept equals the number of WSs (v).

$$w_{max} = v$$

Meanwhile, the minimum number to be allocated (denoted by $w_{min} = l$) equals the number of ML (denoted by l) that fit into a memory page.

$$w_{min} = l$$

The domain-blocks are addressed and mapped according to their physical memory address. In reference to Figure 5.8, the domain-blocks are addressed by the d bits portion.

Table 5.2 Domain-block specific parameters. In addition to Table 4.1

Sign	Description
P_{Size}	Size of a smallest addressable memory-page
CL_{Size}	Size of single cache line
ML_{Size}	Size of single memory line equals to CL_{Size}
DB_{Size}	Size of a domain-block in Byte
DB_i	Identifies a specific domain-block in the memory
l	Number of ML in domain-block
v	Total number of way-sets in the cache
w	Number of way-sets per domain
m	Number of CLs in the cache
k	Number of CLs in each way-set
d	Number of domains supported

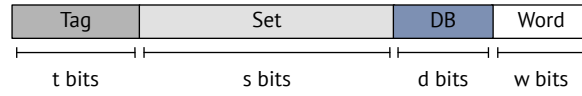


Fig. 5.8 Addressing of domain-blocks in main memory.

5.2.3 Proof-of-Concept Implementation

To deploy the domain-blocks approach and the above-given parameters the actual system specifications have to be taken into account. These include the LLC size, number of domains and the memory page granularity. In the PoC implementation, an exhaustive example of the concept is realized, described and evaluated. It, furthermore, elaborates upon the utilisation of multi-staged page tables, as they are usually implemented in modern MMU architectures.

The employment of domain-blocks implies that all domains follow this mapping pattern. Portions of memory that are not mapped by the domain-block pattern would interfere with the strict separation of WS. However, each domain-block colour must not necessarily be exclusively assigned to an AD. Trusted domains might share a cache-domain in LLC, for example.

Construction of Domain-Block Mappings In the following, an algorithm is shown which outputs a proper translation table based on the domain-block concept.

Translation Table Descriptor The domain-blocks will be mapped to the MC-domains using the 2^{nd} -stage MMU. In the architecture of the experimental platform, the description of the mappings is done in a translation table. This concept is described in Section 2.1.4. The particular description of those page tables is specific to the architectural implementation. In the case of an ARMv7 architecture, there are two descriptor formats supported. Here, the Long-descriptor format³ is used, because 40bit output addresses⁴ are necessary to address the entire PA space of the platform. Here, three levels of descriptors are concatenated in order to provide efficient page table walks⁵. The first level refers to large chunks of memory, whereas the Level 3 descriptors support a page granularity of 4096Byte.

Building the Descriptor Tables The mappings in the *Level 3* descriptors are generated as follows. The input address space represented by the IPA must be consecutive for a proper operation of the OSs utilising them. The output addresses (PA) are generated concerning the proposed domain pattern. In Table 5.3 the key cache parameters are summarized.

In the given example, two domains need to be created in the cache and main memory. According to C_{Size} and CL_{Size} each domain consists of 1024 WSs. By the same token, in main memory, these 1024 WS blocks have to be created. Each addressable page holds 4KiB and therefore 64 CLs with a size of 64Byte. As a result, 16 consecutive Level 3 pages are needed to build a 1024 entry domain-block.

The mappings are generated using the Algorithm 1. The PA space for the main memory starts at address 0x80000000. *MC1*'s IPA base address points to 0x80000000 and for *MC2* at 0xA0000000. The algorithm iterates through the whole address space of the main memory.

Boot Phase Adaption

The proposed mapping separates the shared WSs within the cache into dedicated cache-domains. However, the implementation of the scheme implies challenges, particularly, to the boot sequence of the system.

³Refer to [10, p. B3-1318] for further information on descriptor formats.

⁴ARMv7 is a 32Bit architecture. However, the Large Physical Address Extension (LPAE) enables the handling of larger physical address spaces up to 40Bits [10, p. B1-1159].

⁵Table lookup.

Table 5.3 Platform cache parameters.

Sign	Value	Description
C_{Size}	2MiB	Size of LLC
M_{Size}	2GiB	Size of Main Memory
CL_{Size}	64Byte	Size of single cache line
ML_{Size}	64Byte	Size of a single memory line equals to CL_{Size}
DB_{Size}	1024 cache-lines	Size of a Domain Block (m)
v	2048	Total number of way-sets in the LLC
k	16	Number of CLs in each way-set

Algorithm 1 Generate level 3 translation table

```

 $IPA_1 \leftarrow 0x80000000; IPA_2 \leftarrow 0xA0000000$ 
 $PA_1 \leftarrow 0x80000000, PA_2 \leftarrow 0x80010000$ 
 $Pagesize \leftarrow 0x1000$ 
for  $j = 0; j \leq MainMemory_{size}; j += DB_{size}$  do
  for  $i = 0; i \leq 16; i += Pagesize$  do
     $IPA_{[1,2]} \Leftrightarrow IPA_{[1,2]} + Pagesize$ 
     $PA_{[1,2]} \Leftrightarrow PA_{[1,2]} + Pagesize$ 
  end for
   $IPA_{[1,2]} \Leftrightarrow IPA_{[1,2]} + Pagesize$ 
   $PA_{[1,2]} \Leftrightarrow PA_{[1,2]} + DB_{size} * 2$ 
end for

```

The proposed domain cache mapping must be created during the boot sequence of the system. That means that, in the AMP configuration stage, which is in this case on behalf of u-boot (according to Section 3.1.3), will create the domain mapping rather than the usual identity mapping. This leads into the following *chicken-egg* issue in the boot sequence. U-boot is located within the main memory. The memory mapping for the u-boot code was set by the IPL in advance. Accordingly, u-boot works on identity mapping. This means that, the Linux Kernel images that are loaded would be written to memory in sequential order. Accordingly, the loading of the image needs to respect the new domain-wise memory pattern. As a result, the image loader (u-boot) has to divide the images and configuration files into chunks with the size of the domain-blocks and load each of them to the respective location in memory. For example, the Linux Kernel images have a size of about 3MiB. In the shown example, the domain-block chunks are 64KiB. Accordingly, the images are divided into 47 chunks.

Concluding Remarks

The given concept provides the possibility for separating shared caches by a unique memory mapping technique. It solves the issue of cache contention of competing accesses to common WSs.

The given concept is limited to k-way-set associative LLCs. Other caching concepts such as fully-associative caches do not provide the capability to tag specific cache-lines through a reordering of physical page mappings. In these concepts, the placement and replacement are handled by the cache management, independent of the physical address tags.

In contrast to the proposals mentioned in the related works section 7.4, the characteristics of the system that applies the concept is as follows. Page allocations are only executed during the startup of the system. The page allocation, which is represented in the memory map, is part of the system setup.

5.3 Countermeasure 2: Memory-Map Shuffling

As is shown in the vulnerability assessment and the penetration test (compare with Section 4.2), there is a surface for circumventing the memory protection in an AMP system. Although the assessment only shows particular instances of vulnerabilities and their exploitation, it reveals the major problem to be dealt with: the failure of a primary countermeasure. This can occur either from the design, realization or assumption perspective, as well as many other ways. Possible strategies for encountering this problem are manifold and will be discussed in the following.

In order to prevent attacks, as they are shown in Section 4.2, the memory protection mechanism shall only be configurable from a privileged entity other than the AD's software-stack. The privileged entity is, in the AMP case, software running in the intermediate level, and configures the memory separation. Translation tables must be set up by this intermediate level code and shall be made inaccessible by the CoPE, in this particular example. However, this requires suitable hardware facilities on the MPSoC. Most importantly, in cases where the memory access shall be controlled from the intermediate/hypervisor level, IPMMUs shall be integrated. Sometimes those MMUs are referred to as System Memory Management Unit (SMMU)s [110, 120]. Conceptually, they are intended to work as the second-stage MMUs do, which is the enforcement of IPA to PA address translations/accesses. Nevertheless, the integration of new hardware components is not possible in some situations. For example, it could simply not be available on the market. Or, and this is often the case in the field of

security, the need for such a component has been identified too late for adapting the hardware.

Whereas the former strategy was to extend the protection architecture on the lower system levels, the technical mitigation is also achievable by introducing a further software layer on top, for example, by a Type1/Type2 hypervisor (compare with Section 2.1.2). These instances apply to intercept accesses initiated from the main processors (PE1/PE2 in the given example) and only allow firmware or configurations that are authentic and trusted. Although this is a reasonable approach, it interferes with the original idea of AMP systems to avoid such hypervisor software layers.

Taking the examples into account, here the introduction of a secondary countermeasure is taken into account.

5.3.1 Effectiveness Requirements

Conceptually, the method proposed here should fulfil and answer research question *RQ2* while also concerning the general aim of this research. The general aim requires finding solutions adapting the intrinsic configuration interface of an AMP system, which is the memory map. The countermeasure should implement the concept of a secondary countermeasure.

The security solution shall utilize the capabilities of system memory-maps to reduce the risk of exploitation. It shall encounter a tampering threat on a memory partition by an adjacent processing element that belongs to a different trust boundary (in other words, asynchronous domain). The identified attack potential of 17 (Moderate) must be raised to at least >20 (High).

5.3.2 Mitigation Concept Analysis

Obfuscation is the approach to mitigate the risk of exploitation. The concept aims at increasing the effort of localizing and predicting the targeted structure in the main memory partition of an adjacent domain.

In the history of OS security, the obfuscation of address layouts is a method to increase the effort of exploiting vulnerabilities. Using this technique, it is more difficult for an attacker to determine the location of memory structures. Address space obfuscation, which is often referred to as Address Space Layout Randomization (ASLR), was originally implemented for *user-space* applications [168] and has been extended

to the kernel-space, for example, in Linux [37]. It added an artificial diversity of the memory locations of the applications *Stack*, *Heap* and linked libraries and positions within a process's address space. Thus, the exploitation of buffer-overflow and format-string vulnerabilities have become harder.

By the same token as ASLR, in this approach the identity mapping (compare with Figure 5.9) between the IPA and PA address mapping will be changed into an unpredictable mapping. The utilization of the stage-two mapping will be handled as it is introduced in the Domain-Block concept in Section 5.2.3.

The concept aims at placing the physical addresses in such a manner that without the knowledge of the mapping table an adversary cannot reconstruct the entire memory structure of an adjacent memory partition. That means, that after the access control breach, the attacker is able to jump to and access every position in main memory. Nevertheless, at PA level, it cannot be differentiated to which MC-domain the memory pages belong. Furthermore, the order of pages is not sequential after the obfuscation. According to the ToE, Figure 5.9 visualizes this principle. To summarize, the obfuscation takes effect in two dimensions: first, the page assignment and second, the sequential order of the pages.

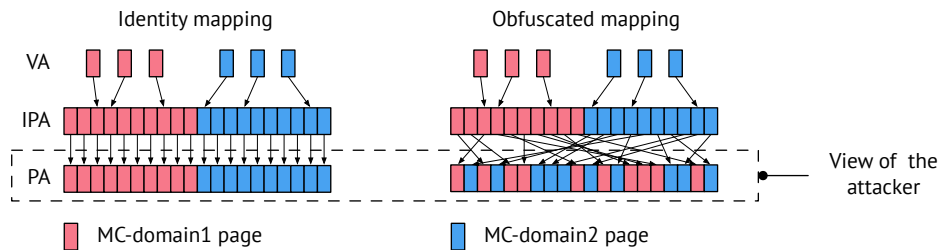


Fig. 5.9 Principle of randomized memory assignment.

Shuffling Procedure

The core of the obfuscation concept is the algorithm to produce the permutation of the address mappings. The procedure of randomizing the address space can be compared to shuffling of a deck of cards. Therefore, in order to transport the overall approach, the shuffling algorithm by Fisher and Yates [48] is chosen. The Fisher-Yates shuffle is simple and fits well to produce random permutations of finite sets. In this particular case, the finite set is the translation table which was previously created by the initialization process during the start-up of the system. The translation table is

denoted as a finite set TT of mapping entries E .

$$TT = \{E_1, E_2, \dots, E_n\} \quad (5.1)$$

Each entry redirects an intermediate (IPA) input address to a corresponding output address range. We assume TT is initialized with an identity mapping, which means each intermediate address directs to the equal to the physical address $IPA = PA$. The entries of the set would then be arranged as follows:

$$TT = \{PA_{0x000000001}, PA_{0x0000000040}, \dots, PA_n\} \quad (5.2)$$

The Fisher-Yates algorithm is shown in Algorithm 2. It iterates through TT and swaps the entry in the current position with a random position. The random position is determined by a randomization function which draws values out of a specified range.

Algorithm 2 Memory-Map Shuffle

```

for all  $TT[i]$  do
   $random \leftarrow$  random number such that  $0 \leq random \leq range$ 
  swap  $TT[random]$  and  $TT[current]$ 
end for

```

System Requirements

However, beyond identifying a proper permutation procedure the concept builds on certain aspects relevant to the target environment. Architectural or technical constraints are relevant as well as procedural prerequisites.

Boot Process Integration. The permutation procedure must be integrated into the boot process of the AMP system. This is motivated by the fact that the intermediate mappings are generated and applied during the startup. Therefore, the system boot process introduced in Section 3.1.3 will be advanced by the permutation process. In Figure 5.10, an overview of the causal dependencies of the concept is shown. After the boot PE has initialized the hardware, the AMP is depended on the setup of the second stage memory-map that is generated. In this case, first is the identity mapping. The shuffling of the mappings is invoked by the *permutation procedure()*. This procedure retrieves random numbers in the defined range of the mapping table and reorders with them the translation table (as it is shown in the Fisher-Yates shuffle example previously in Section 5.3.2.). Having finished that procedure and set up the translation

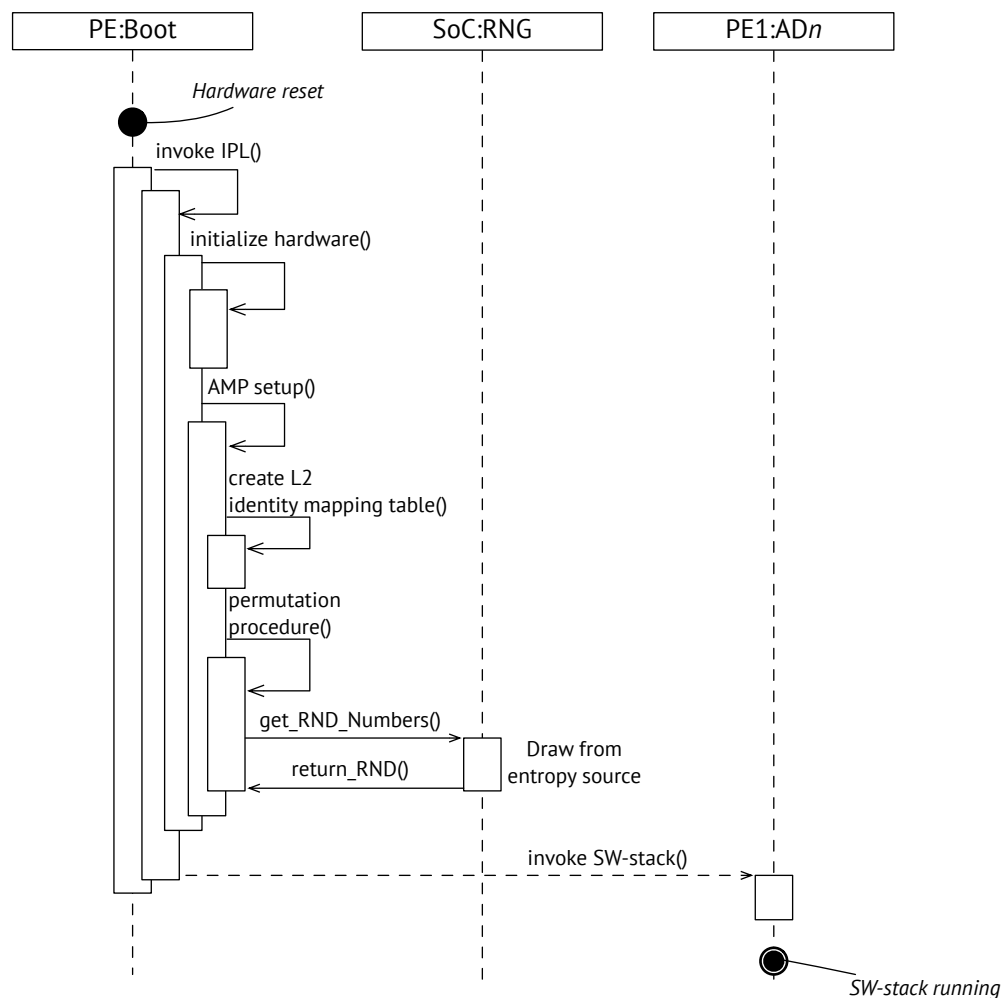


Fig. 5.10 Integration into the boot process.

tables properly, the particular software-stacks of the ADs can be invoked. During their startup, the first-stage memory-maps will be created on their behalf.

Random Number Generation. One of the key elements of the shuffling algorithm or the permutation algorithm is a suitable Random Number Generator (RNG). As shown in Listing 2, a discrete random number from a specified range ($0 \leq \text{randomNumber} \leq \text{Range}$) is drawn. It is assumed that a cryptographic secure RNG is providing sufficient entropy. The required entropy depends on the granularity of page mappings of the system, in other words, the total number of entries in TT . Furthermore, it is required that the generated numbers are still non-biased after truncating them to the specified number range. The Fisher-Yates shuffle sufficiently fits to introduce

and transfer the core idea of the memory mapping approach. However, the practical implementation of a non-biased random permutation isn't trivial for certain aspects. As an example, to fit the random numbers into a specific range a *modulo* operation could be used. It has been shown that this will lead to a biased output.

Alternatively, only those numbers could be drawn which are in the specified range. This means it has to wait until the RNG draws a value which suits the targeted range. Practically, this could substantially contradict to startup time requirements of the ToE.

Performance plays an important role since this approach will be integrated into a critical timing environment. Every time the system is reset the memory mapping will be randomized. Therefore, the algorithmic complexity must be kept to a minimum, so the startup phase of the device is not significantly delayed.

Effectiveness Metrics

Overall, the effectiveness of the given approach relates to the ability of the attacker to guess or determine the position of randomly created data. In the evaluation of address space randomization, three aspects are taken into account [13, 114]:

Coverage: All areas of the memory layout must be randomized to defeat attackers. Known attacks on ASLR have shown that non-randomized memory areas can be used to circumvent the obfuscation.

Entropy: The entropy is the metric to express the grade of uncertainty of the randomized set. Therefore, the range of entropy must be as high as possible. The granularity of the shuffled memory pages plays an important role.

Relocation Frequency: The relocation frequency is important in determining how much time an adversary has to break a particular randomized memory layout.

Coverage. The proposed memory-map shuffle concept could be applied to the entire system-memory space that is addressable by the second-stage MMU or SMMUs of a MPSoC. However, not all of the addresses can be chosen dynamically. For example, the configuration registers that are mapped to a particular area to the address space are statically defined. As a consequence, the shuffling of addresses is only feasible only to the addresses that represent the main-memory partitions within the address space.

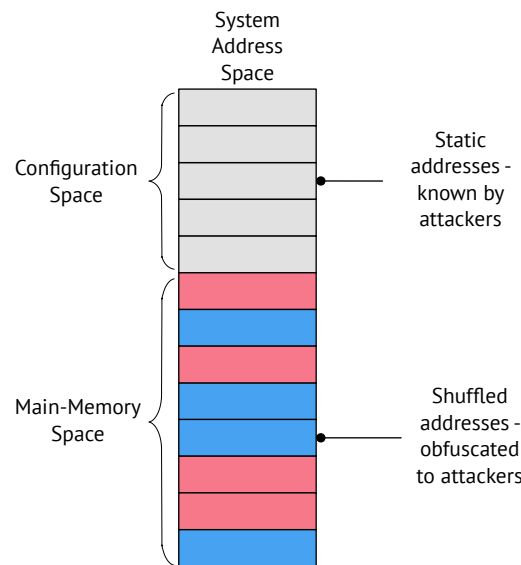


Fig. 5.11 Coverage of the main memory shuffling.

Entropy. This metric is an important factor in determining the probability of success by a certain amount of attack attempts. In Table 5.4, the symbols for calculating the probability are stated. The bits of entropy will be derived from the address space that is to be shuffled. An influencing aspect here is the granularity of the pages that are handled within the translation table. Most commonly, the smallest page size is 4KiB. This is particularly true for the experimental platform utilized throughout this thesis. However, this is to be treated as architecture dependent.

Table 5.4 Entropy and probability symbols.

Sign	Description
E	Entropy bits for the shuffled address space
α_f	attack attempts for fixed randomization
α_c	attack attempts for changed randomization
P_{Size}	Size of a addressable memory-page
M_{Size}	Size of memory to be shuffled
N	Number of translation table entries
P_f	Probability for fixed randomization
P_c	Probability for changed randomization

In order to determine the entropy bits, the number of entries in the translation table has to be calculated. Here, the size of a single addressable memory page, denoted by P_{Size} , and the size of the memory area that is to be shuffled, denoted by M_{Size} , must be taken into account. N denotes then the number of entries by the following equation:

$$N = \left(\frac{M_{Size}}{P_{Size}} \right)$$

The resulting entropy bits (denoted by E) are calculated by:

$$E = \log_2 N$$

Relocation Frequency. In the considered ToE, the relocation frequency is equal to the reset cycle of the device. Since the shuffling is executed on the startup, a relocation can only be done when the system is restarted. A relocation during runtime is infeasible because the system would need to be halted. Then, the IPA memory must be copied to a buffer area and copied back after the mapping has been applied to the PA. In such cases, a restart of the whole system would be more efficient and reliable.

Probability and Attack Attempts. According to Shacham et al. [152], there are two ways to approach the probability with regards to the relocation frequency: either the randomized set is fixed during the attack or the randomization changes with each attack attempt. The former case describes a simple random sampling without replacement to determine the probability. Whereas, the latter one, is random sampling with replacement. The reference equation is shown in the following⁶:

$$P = 1 - \frac{N - n}{N}$$

Fixed Randomized Set. The following equation applies for the probability (denoted by P_f) of a fixed randomized set.

$$P_f = \frac{1}{2^E}$$

The number of attack attempts given this probability is:

$$\alpha_f = 2^{E-1}$$

⁶ P denotes the probability, N the population and n the sampling size.

Changed Randomized Set. The following equation applies for the probability (denoted by P_c) of a changed randomized set.

$$P_c = \frac{1}{2^E}$$

The number of attack attempts given P_c is:

$$\alpha_f = 2^E$$

Practical Implications. Given these two ways of predicting the attack attempts, the question arises which of these scenarios apply to the considered environment. It is assumed that the adversary intends to tamper with the control flow of its target. It is considered that the adversary ends up in two situations after guessing the memory mapping. Either, he succeeds and hits its target, or he tampers with the wrong memory locations. Here, it must be presumed that the target has an inconsistent or unstable control flow which results in unpredictable behaviour. The system has to be rebooted which will result in a different memory map. As a result, practically, the probability and attack attempts shall be calculated according to a random sampling with replacement.

5.3.3 Case Study

In the following, the above-described metrics are evaluated by giving practical examples. The intention is to provide practical examples of entropy bits compared to common page sizes and main memory sizes.

This includes the parameters of the experimental platform as it is described in Section 3.1.3. It implements the ARMv7 system architecture specification. Essentially, the architecture supports two-page sizes on the level three descriptors, which are either 4KiB or 2MiB. On the higher levels of the descriptors, for example, 1GiB pages are supported.

In Chart 5.12 the page sizes: 4KiB, 2KiB, 1KiB, 2Mib and 1GiB are examined. This is correlated to main memory sizes ranging from 1GiB up to 128GiB. Although, 2KiB and 1KiB are not common the effect of a larger number of translation table entries to the entropy bits become visible.

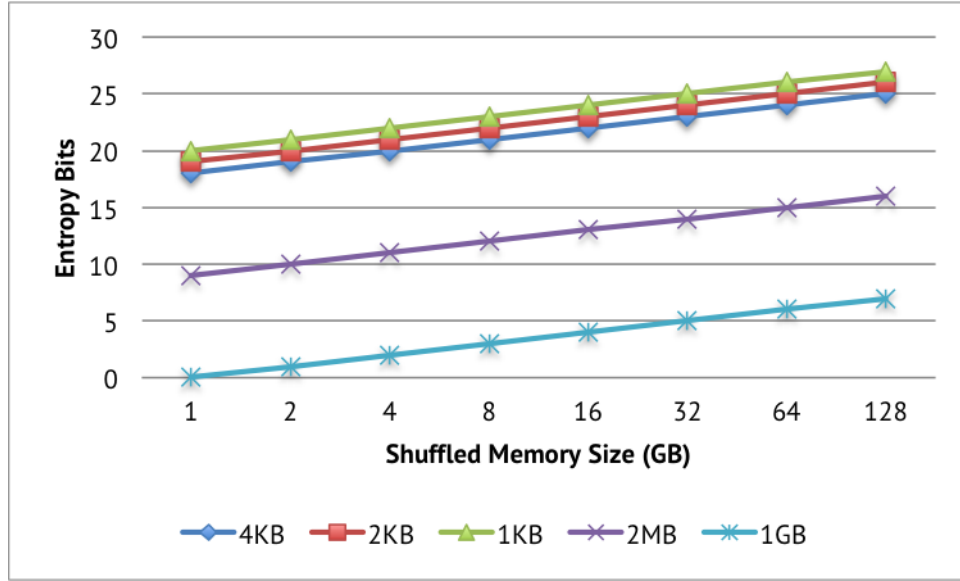


Fig. 5.12 Comparison of page size, shuffled memory size and the resulting entropy bits.

Focusing on the properties of the experimental platform, which is 4KiB P_{Size} and 4GB M_{Size} , an entropy of 20 bits is reached. Assuming a M_{Size} of 128GB, the entropy would rise to 25 bits whereas the more coarse-grained page size of 2MiB end up with 11 bits of entropy for 4GB main memory and 16bit for 128GB.

Discussion

By applying the random permutation on the intermediate physical address mappings, the physical memory structure is obfuscated. It is obvious that exploits such as referenced in the vulnerability assessment would fail. However, adversaries adapt to the newly introduced circumstances and try to de-obfuscate the memory map. Referring to crypto-analysis, a reasonable approach to evaluate the effectiveness of this kind of statistical security control is used to estimate the effort needed to brake it. In general, we assume two approaches to compromise a permuted address mapping: either the attacker scans through the whole main memory for a page they are looking for or by applying statistical analysis on the permutation procedure. The former approach makes it necessary to assume that the attackers are able to scan through the whole main memory space. Furthermore, they need an evaluation function that determines whether or not the current scanned page is the one they were looking for. This is what we also described in our threat scenario. However, the adversary now has to deal with

the fragmentation of binary patterns. By applying this brute force attack, the attacker needs to scan half of all left pages on average to find the next designated page.

State-of-the-art ASLR mechanisms gain up to 16-bit entropy for 32-bit architectures and 28 to 40 bits of entropy in 64-bit architectures [114, 152]. However, this depends on the particular OS and the way in which all code sections are linked into the virtual address space.

Furthermore, the approach of ASLR is applied to the OS-level as well. Current Kernel Address Space Layout Randomization (KASLR) mechanisms provide from 6 bits entropy (Linux) over 8 bits (OS X) to 13 bits (Windows) [88]. As a result, the herein proposed solution is comparable to solutions applied to higher systems layers. However, the most important advantage, is the independence to other solutions applied to the higher layers. That means, the combination of all three obfuscation techniques increases the exploitation effort for adversaries.

Compared to these state-of-the-art (K)ASLR implementations, the intermediate obfuscation is even more powerful. ASLR randomizes the relative offset of memory segments (such as code, text, etc ...) of an application according to a base address. After the adversary guessed this offset successfully, he can exploit the structure by assuming a sequentially ordered memory. In contrast, memory-map shuffling reorders every memory page so that no sequential order of memory structures can be expected.

The concept proposed here is not considered to introduce significant performance degradation. The mechanism is integrated into the second stage address translation which is utilized for every address access either way. However, due to the scattered, non-sequential, order of memory pages, the cache misses in the Translation Lookaside Buffer (TLB) are expected to be higher compared to a system utilizing page-tables referring to 2MiB pages instead of 4KiB, for example. This is also discussed in the Section 6.1.2 in the evaluation the proposed Domain-Block mapping.

5.4 Summary

This chapter elaborated upon the treatment of risks that were identified in the previous chapter. It, therefore, completes the consideration of security patterns defined throughout this work. For that purpose, here, an exploitation of prevention mechanisms and a denial-of-service mitigation have been proposed. Based on the generalized findings of the vulnerability assessment, the concept of differentiating countermeasures in *primary* and *secondary* has been proposed. Whereas a primary countermeasure

defeats a particular vulnerability, a secondary countermeasure aims at preventing from unknown vulnerabilities or exploitation attempts.

Taking these concepts into account, a *domain-block* memory-mapping has been introduced. This concept realizes an effective mitigation against denial-of-service attacks against shared WSs in a LLC. It realizes a primary countermeasure by applying the proposed mapping ADs work on private memory-chunks (domain-blocks) in the shared cache. An enforced eviction of cache-lines by an adversary is prevented with this technique. A limitation of the approach is that the cache will statically be divided between the ADs. For applications seeking for best-effort performance, this might have an impact.

Furthermore, in this chapter, a concept to mitigate the effects of breaching the hardware memory protection is proposed. Here, a secondary countermeasure preventing the exploitation has been designed. The obfuscation of the intermediate address mapping is based on the introduction of random permutations of a normal, continuous memory page arrangement. The so-called "memory-map shuffling" increases the effort to re-interpret the memory structures. The shuffled-memory map is transparent to higher layers in the system architecture. Revisiting the system properties of the experimental platform, the proposed approach reaches 20 bits of entropy, which is comparable to common ASLR implementations of 32-bit system architectures.

Security Evaluation

Contents

6.1	Domain-Block Memory Mapping	174
6.1.1	Effectiveness Assessment	174
6.1.2	Evaluation of the PoC Implementation	176
6.1.3	Residual Risk Analysis	181
6.2	Memory-Map Shuffling	183
6.2.1	Effectiveness Assessment	183
6.2.2	Residual Risk Analysis	186
6.3	Comparison to Hypervisor-based System Architectures .	188
6.3.1	Attack Potential: Cache-Thrashing	189
6.3.2	Attack Potential: Tamper with Memory of Adjacent OS Guest	191
6.4	Summary	192

If it's provably secure, it's probably not.

Lars Knudsen

Proving a system's security can be cumbersome or even contradictory. This means that with particular efforts, it must be shown whether the protection is adequate based on assumptions and anticipations. There will be other, presently unknown, circumstances that make the assumptions obsolete, so the protection mechanism is ineffective.

In this chapter, the security evaluation is conducted. The evaluation is based on the process introduced in Section 2.2.2. For both proposed countermeasures, an analysis and evaluation of the effectiveness is elaborated upon. Furthermore, the case study is revisited by determining a residual risk level based on the findings in the effectiveness evaluation.

In addition, the chapter features an AP based comparison of a system architecture that implements a hypervisor to separate system domains.

6.1 Domain-Block Memory Mapping

6.1.1 Effectiveness Assessment

Effectiveness Analysis

The proposed Domain-Block mapping seeks to implement a primary countermeasure. As a result, it will mostly affect the security property availability. Furthermore, since it solves the problem of commonly used way-sets, the change of *Scope* is focused as well. As it is shown in Table 6.1, the Modified Vulnerability Score is rated at 1.9 (Low).

Modified Vulnerability Score

Rationale. Most of the score factors remain unaffected. Only the availability aspect is lowered from the score *high* down to *low*. This is motivated by the fact that the shared resource (LLC) is isolated from domain to domain. In accordance with the penalty metrics given in 4.1.1 the following achievements have been made.

The accumulated $t_{\delta_DoS_{penalty}}$ is directly avoided due to the separated way-sets. As a result, there is no surface for interference in this regard. Accordingly, the impact on the execution time is prevented as well. However, there might be infrastructure such as communication interfaces between the cache levels that are still used concurrently.

Table 6.1 CVSS base score of the domain-block concept.

Base Score Type	Rating	Score
Attack Vector (AV)	Local	
Attack Complexity (AC)	High	
Privileges Required (PR)	High	
User Interaction (UI)	None	
Scope (S)	Unchanged	
Confidentiality (C)	None	
Integrity (I)	None	
Availability (A)	Low	
Base Score	Low	1.9

Attack Complexity. The proposed concept does not mainly affect the complexity of exploiting the common shared cache. Therefore, the AC metric remains at *High*. However, on a lower level of detail, there might be other ways to degrade the access performance of an adjacent domain in the system.

Scope. Changing the scope is not possible after applying the proposed concept. By the fixed assignment of way-sets in the LLC, an adversary can only affect way-sets that belong to its domain. As a result, the *Scope* metric is rated at *Unchanged*.

Confidentiality. The confidentiality of information is not affected.

Integrity. The integrity of information or code is not affected.

Availability. Impacts to the availability are rated at *Low*. Particularly, the measurements in the penetration test revealed that the proposed mapping takes effect. However, the impact of the DoS test loops are still observable, but significantly limited. Therefore, it is unreasonable to rate the availability at *None*.

Effectiveness Evaluation

The evaluation of the effectiveness of the given domain-block approach refers to the vulnerability score gained in the evaluation of the corresponding attack in Section 4.1. The effectiveness of the approach is represented by the delta of the exploitability severity measured with and without the applied concept. The results are summarized in Table 6.2.

Table 6.2 Effectiveness Domain Block concept.

Base Score Type	Countermeasure	Vulnerability
Attack Vector (AV)	Local	Local
Attack Complexity (AC)	High	High
Privileges Required (PR)	High	High
User Interaction (UI)	None	None
Scope (S)	Unchanged	Changed
Confidentiality (C)	None	None
Integrity (I)	None	None
Availability (A)	Low	High
Exploitability Base Score	Low (1.9)	Medium (5.3)

6.1.2 Evaluation of the PoC Implementation

Experiment Setup

The experiment seeks to demonstrate the effectiveness of the proposed domain-block mapping concept. Hence, the experiment implies two tests. The first shows the effect of a cache thrashing attack as it is introduced in Section 4.1. In the second, the mitigation method, as it is shown in Section 5.2 is tested. Two hypotheses are formulated and tested accordingly.

Generally, the experiments measure a delta between the execution time of a program with and without the applied concepts. The resulting *impact-delta* is used as evidence to test the hypothesis (**Hypothesis_{e2}**).

Hypothesis_{e2} The applied domain-block mapping mitigates the former cache thrashing attack.

H₀: The applied countermeasure shows no impact on the delta measurement.

H₁: The impact-delta is decreased by the application of the domain-mapping.

In Table 6.3, the additional symbols for the measurements are defined. This advances the contents given in Table 4.5.

In order to determine the delta between the identity mapping and the domain-block mapping the according difference is calculated by Equation 6.1.

$$\left(\frac{\sum_{i=0}^{k=10000} n_1(i) - n_2(i)}{k} \right) - \left(\frac{\sum_{i=0}^{k=10000} t_1(i) - t_2(i)}{k} \right) i, k \in \mathbb{N} \quad (6.1)$$

Table 6.3 Measurement symbols.

Name	Equation	Description
s_i	$s_i \in \mathbb{N}$	CPU cycle count value at the start of iteration.
e_i	$e_i \in \mathbb{N}$	CPU cycle count value at the end of iteration.
k	$k \in \mathbb{N}$	Number of iterations
δ_t	$\left(\frac{\sum_{i=0}^k e_1(i) - s_1(i)}{k} \right) i, k \in \mathbb{N}$	Delta of DoS/thrashing measurements
δ_n	$\left(\frac{\sum_{i=0}^k e_1(i) - s_1(i)}{k} \right) i, k \in \mathbb{N}$	Delta of normal duty measurements
δ_{db}	$\delta_t - \delta_n$	Delta reflecting domain block mapping
δ_i	$\delta_t - \delta_n$	Delta reflecting identity mapping
s_n	-	Standard Deviation of δ_n
s_t	-	Standard Deviation of δ_n

Domain Block Mapping. To evaluate the effectiveness of the domain-block mapping, the measurements are repeated with a system that applies the domain-block mapping concept. Here, evidence to test *Hypothesis_{e2}* is gained. The test execution is similar to the previous test. The results are visualized in a graph shown in Figure 6.1 and summarized in Table 6.4. The normal execution of the *measure-loop* reveals a mean cycle count (δ_n) of 3,599 (4 cycles). By invoking the *DoS-loop*, the mean cycle count (δ_t) is 8,911 (9 cycles). Consequently, this results in a DoS impact of about 247,55%. The according δ_{db} is in this case 5,321 cycles.

As a result, the measurements prove evidence that H_1 of *Hypothesis_{e2}* applies. Due to the application of the domain-block mapping, the impact of the *DoS-loop* is decreased.

However, the results still show a measurable impact even with the applied domain-block mapping. Practically, this might be caused by the competing usage of the interfaces and facilities within the caching infrastructure.

Overhead Evaluation. The previous evaluations show that the proposed approach is effective against DoS attacks in AMP-based MC-systems. Despite the advantages of the separability of the LLC, further performance aspects are evaluated here. Accordingly, the approach is compared to the memory access latency of single memory access. Furthermore, the latency for memory accesses in consecutive memory areas in copy

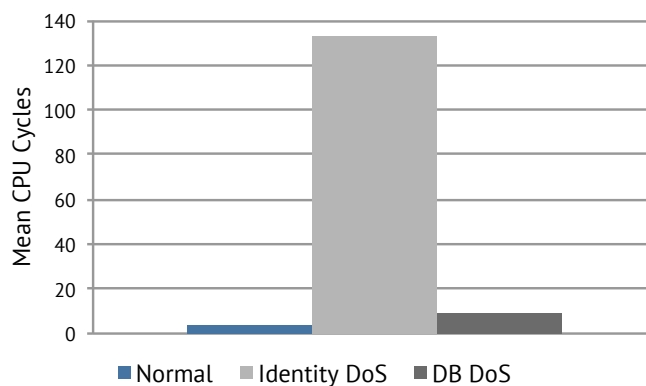


Fig. 6.1 Comparison of identity and DB mapping.

Table 6.4 Overview of the measurement results.

Mapping	Impact (%)	CPU Cycles (δ)	Std. Derivation (s)
identity	-	3,602 (δ_n)	0,022 (s_n)
DB	-	3,599 (δ_n)	0,021 (s_n)
identity DoS	3686,90%	132,803 (δ_t)	2,265 (s_t)
DB DoS	247,55%	8,911 (δ_t)	2,062 (s_t)

operations will be measured. This shall show how the approach behaves in applied scenarios.

TLB Refreshing. The introduction of the proposed memory mapping will introduce overhead by retrieving the mapping entries from the MMU translation tables. This might be caused by additional table walks. A full translation table lookup is called a translation table walk. As mentioned previously, the MMU translation tables are concatenated tables, where each level describes a finer-grained amount of data, from the higher to the lower levels. Using an identity memory mapping, the granularity of *Level 2* block descriptors are sufficient to build a proper system mapping. According to the domain block mapping, *Level 3* descriptors which map to the 4KiB pages need to be used rather than the 2MiB blocks in the second level. This implies that a lookup for a physical output address needs to walk through all descriptor (Levels 1, 2, 3). The number of entries in the Levels is dependent on the amount of memory to be addressed by the MMU.

As mentioned above, the mapping method in the second stage MMU level implies three table walks to convert the IPA input address into the PA output address. In order to quantify the cost of the additional table walk, the latency of the *measure-loop* has been observed with the original 2 MiB *Level 2* mapping and the domain block mapping which reside in the *Level 3* translation table. The results show that the DB mapping method does not add any performance overhead to the memory access if a single WS is considered. The mean CPU cycles remain at 3,602 in identity mapping and 3,599 using the DB mapping.

In addition to the table walks, the MMU implements a cache for its recently used translation entries. This cache is commonly referred to as a TLB. Despite that the size of the TLB is specific to the implementation, the experimental platform's TLB caches 512 translation descriptors. If a normal identity system-mapping is assumed and the experimental platform, utilising Level 2 descriptors for 2MiB blocks, there is the maximum number of 2048 descriptors to be stored in the TLB. In contrast to the DB mapping method which uses fine-grained Level 3 descriptors, there are 1048576 (size of main memory divided by the size of page mapped by the Level 3 descriptor) translations. Accordingly, the probability of refreshing the TLB is much higher than the DB mapping compared with the normal method. The result would be a certain overhead introduced by the increased number of TLB refreshments.

Particularly in this architecture, the TLBs store up to 512 translation descriptors. In the conducted measurements, only 32 different memory addresses are utilised. Accordingly, the TLB is large enough to cache those descriptors in this example. To measure the impact of frequent TLB refreshes, the number of the iterated cache-lines in the *measure-loop* have been increased to 512 on each MC-domain. This forces the TLB to refresh TLB descriptor entries.

The results quantify the impact to access latency caused by refreshing the TLB. The introduction of the domain mapping increases the mean access latency of the *measure_loop* at about 2.52 percent.

Memory Bandwidth. Practically, driver information systems sometimes tend to transfer large amounts of data in memory. This might be graphics data or media files which are processed. Therefore, it is evaluated as to how the memory bandwidth behaves when large amounts of data are copied. Figure 6.2 shows a comparison of the mean CPU cycles consumed by copying a particular amount of data. It analyses the overhead for large data operations. The data block is contiguously allocated in the intermediate address space (IPA). The results show that the bandwidth of the identity

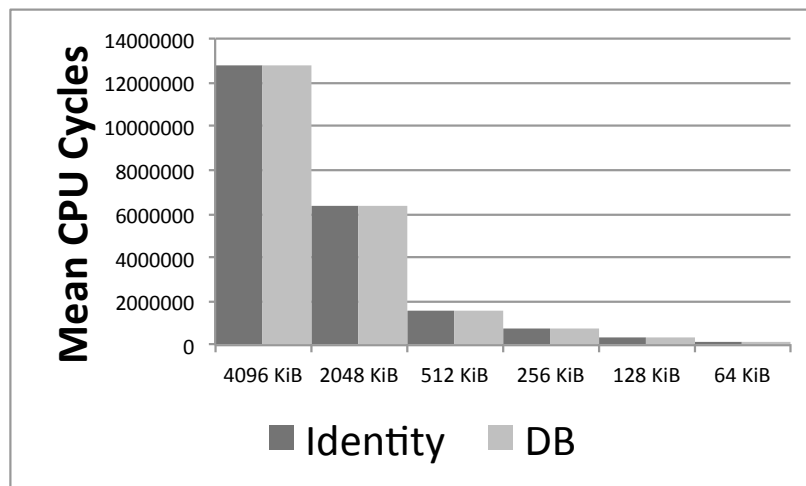


Fig. 6.2 Memory bandwidth comparison.

and the DB mapping is barely at the same level. In fact, the domain block approach has no influence on the transfer of contiguous memory areas.

Environmental Vulnerability Score

Due to the character of the considered attack and vulnerability the environmental requirement of availability is concerned. For the sake of completeness, the other requirement metrics are listed as well. The environmental metric values are influenced by the case study risk analysis shown in Section 3.3.3. As a consequence, the modified score resulting from this metric does only apply for this particular case study.

The result of the environmental consideration of the CIA security requirements is shown in Table 6.5. The case study reveals that a *Medium* severity rating goes for integrity and availability. Confidentiality is not concerned and therefore is rated at *Low*. A medium rating will not influence, the CVSS. Since there is no impact to the confidentiality with the given vulnerability, the *Low* rating does not affect the resulting score.

Table 6.5 CVSS Environmental score of the Domain-Block mapping.

Environmental Metric	Rating	Multiplier
Confidentiality Requirement	Low	0.5
Integrity Requirement	Medium	1
Availability Requirement	Medium	1
Env. Score Countermeasure	Low	1.9
Env. Score Vulnerability	Medium	5.3

Concluding Remarks and Limitations

The evaluation shows the negative performance impact of the deliberate interference of shared LLCs. By utilising the domain-block mapping, it is possible to mitigate these effects substantially. Furthermore, it has been shown that the performance overhead for domain blocking is negligible. However, there is no total freedom of interference between the two domains, which is demonstrated by the still measurable impact even with the applied domain-blocks. Here, one of the limitations of the approach becomes visible. The mapping only controls the separation of WSs in a cache. If the cache implements shared interfaces to transfer data between the cache levels, there is no possibility to control the competing accesses with the proposed approach. Hence, this surface for interference depends on the particular hardware implementation and has to be considered. A reasonable consideration would be the WCET requirements of applications ran by the target. If the worst-case is still acceptable, which means including the maximum impact, then the solution is applicable. As a result, the measurements show the worst-case scenario.

6.1.3 Residual Risk Analysis

Modified Attack Potential

The resulting AL corresponding to the AP of rated at 26 is 1. Compared to the initial AP consideration, the AL was 4.

Table 6.6 Attack: PE_i disrupts PE_j access to LLC with applied Domain-Block mapping.

Factor	Level	Value
Elapsed time	≤ 3 Months	10
Expertise	Expert	6
Knowledge of system	Restricted	3
Window of opportunity	Unnecessary/unlimited	0
Equipment	Bespoke	7
AP: Beyond high		26

Rationale.

Elapsed time: The investigation and reconnaissance are expected to take approximately three months. Following the levels estimated for the other factors, it will take an effort to gain knowledge of the internals of the caching infrastructure.

Expertise: A potential attacker needs to apply expert knowledge to mount the attack. MPSoCs or generally the embedded area deals with highly customized non-standardized architectures. The re-engineering of concepts and designs is subject to rigid preparation. This remains unchanged.

Knowledge of system: With a countermeasure applied, the attack needs to investigate and re-engineer on a deeper level of detail. It is considered, that not all information is publicly available.

Window of opportunity: This is considered to be unlimited since it was assumed that the attacker already found its way into the system. This remains unchanged.

Equipment: DoS is expected to be caused in a state where the device operates normally. The effects of an attack might be observable with specialised measuring software. Since some of the details are not considered to be documented, the adversary needs to conduct bespoke equipment to reveal the internals.

Residual Risk Definition

This section intends to revisit the risk analysis based on the case study of the Driver Information System which is introduced in Paragraph 3. Based on the risk of the

hypothesised attack (**Hyp-Attack1**: PE_i disrupts PE_j access to LLC), the residual risk is evaluated involving the findings of the modified attack potential. Furthermore, the environmental aspects of the security requirements are briefly evaluated in the following paragraphs.

Residual Risk: Driver Information System. Considering the AP of 26 (*Beyond High*), the resulting AL on low level. The effort to break the vulnerability is high, and therefore, the probability that an adversary takes this path is considerably small. Taking the *Impact Severity* from the case study into account, the resulting residual risk level is 1. The risk level is, therefore, lowered by two levels from a medium (4) down to low (1).

In the given setup, the low risk can be treated as acceptable. However, if the Impact Severity rises to higher values, a risk mitigation might be required. For example, considering an AL for 1 risk level up to 5 is still possible¹. A feasible scenario in the automotive environment would be a system that integrates safety-relevant functions, for example, for ADAS or even automated driving systems. In such cases, the system design requires countermeasures on other system levels.

Table 6.7 Domain-Block Mapping: residual risk analysis results.

Threat	Attack	AP	AL	Impact Severity	Residual Risk
DoS	PE_i disrupts PE_j access to LLC	26	1	C2 S2	1

6.2 Memory-Map Shuffling

6.2.1 Effectiveness Assessment

Effectiveness Analysis

The proposed countermeasure is effectively an exploitation prevention mechanism. It facilitates the concept of a secondary security countermeasure, as is conceptualised in Section 5.1.4. As a consequence, the effect to the particular metrics applies mostly to the *attack complexity* and to the impacted security objectives. These include the confidentiality, and mainly, the integrity. In Table 6.8, the modified vulnerability score is shown.

¹Compare with the security risk level Table 2.4

Table 6.8 CVSS base score of the memory-map shuffling concept.

Base Score Type	Rating	Score
Attack Vector (AV)	Local	
Attack Complexity (AC)	High	
Privileges Required (PR)	High	
User Interaction (UI)	None	
Scope (S)	Unchanged	
Confidentiality (C)	Low	
Integrity (I)	Low	
Availability (A)	Low	
Base Score	Medium	5

Modified Vulnerability Score

Rationale. Score types including *AV*, *PR*, *UI*, *A* are not affected by the application of the memory-map shuffling concept. Therefore, they are not elaborated upon in the following rationale.

Attack Complexity. The complexity for an attacker to successfully mount an attack against the vulnerability significantly rises. There is a need for a target-specific reconnaissance to de-obfuscate the shuffled memory mapping or at least portions of it. The shuffled memory mapping changes every rest-cycle since it is created at every startup of the system. Transferring the de-obfuscated mappings to other targets is prevented by this.

Scope. The proposed approach does not influence the ability to change the scope (such as elevation of privileges) of the target. Due to its nature, the countermeasure does not fix the root-cause of the memory access vulnerability.

Confidentiality. The disclosure of information is also possible although the adversary has increased the effort to find the targeted information. Reading the data, then, does not further impact the system. In other words, the de-obfuscation could be done offline with increased resources and without interfering with the system. As a result, the rearrangement of data is not sufficient to protect from information disclosure with the memory-map shuffling technique.

Integrity. The impact on the integrity of either information or the control flow is rated as *Low*. The attacker can still modify the content of the memory. Therefore, a rating of *None* is not reasonable. However, due to the obfuscation of the exact position of the targeted data, it is not controllable by the attacker. To identify the position is subject to in-depth rest-cycle reconnaissance.

Effectiveness Evaluation

Based on the Modified Vulnerability Score (compare with Table 6.8) the effect of the memory-map shuffling is conducted and justified in the following. The Base Score of the discovered vulnerability is rated at 8.1 (High). The applied countermeasure limits the score down to 5 (Medium).

Table 6.9 Effectiveness memory-map shuffling countermeasure.

Base Score Type	Countermeasure	Vulnerability
Attack Vector (AV)	Local	Local
Attack Complexity (AC)	High	Low
Privileges Required (PR)	High	High
User Interaction (UI)	None	None
Scope (S)	Changed	Changed
Confidentiality (C)	Low	High
Integrity (I)	Low	High
Availability (A)	Low	Low
Base Score	Medium (5)	High (8.1)

Effectiveness Score

Environmental Vulnerability Score. Due to the character of the considered attack and vulnerability, the environmental requirement of availability is a major concern. For the sake of completeness, the other requirement metrics are listed as well. The environmental metric values are influenced by the case study risk analysis shown in Section 3.3.3. As a consequence, the modified score resulting from this metric does only apply for this particular case study.

The result of the environmental consideration of the CIA security requirements is shown in Table 6.10. The case study reveals that a *Medium* severity rating on integrity

and availability. Confidentiality is not to be concerned and therefore rated as *Low*. A medium rating will not impact, the CVSS. The analysed vulnerability has an impact to the confidentiality of the ToE. Accordingly, the multiplier affects the CVSS. For the countermeasure this means that, the resulting CVSS drops from 5 to 4.5. Referring to the vulnerability rating, the rating drops from 8.1 to 7.5.

Table 6.10 CVSS Environmental score of the Memory-Map Shuffling.

Environmental Metric	Rating	Multiplier
Confidentiality Requirement	Low	0.5
Integrity Requirement	Medium	1
Availability Requirement	Medium	1
Env. Score Countermeasure	Medium	4.5
Env. Score Vulnerability	High	7.5

6.2.2 Residual Risk Analysis

Modified Attack Potential

The resulting AL corresponds with the AP of rated at 27 is 1. Compared to the initial AP consideration, the AL was 3.

Table 6.11 Modified Attack Potential: PE_i is tampered with by adjacent PE with applied Memory-Map Shuffling.

Factor	Level	Value
Elapsed time	> 6 Months	19
Expertise	Multiple experts	8
Knowledge of system	Restricted	3
Window of opportunity	Unnecessary/unlimited	0
Equipment	Bespoke	7
AP: Beyond High		27

Rationale. By applying the random permutation at the intermediate physical address mappings, the physical memory structure is obfuscated. Exploits like those referenced in the threat scenario would fail. However, adversaries would adapt to the newly introduced circumstances and try to de-obfuscate the memory map.

Elapsed time: The investigation and reconnaissance are expected to take approximately one week. This is in accordance to the levels estimated for the other factors. Due to the complexity, by gaining knowledge about the system, the estimated time to mount further attacks on the system is rated above six months.

Expertise: A potential attacker needs to apply expert knowledge in a certain field to de-obfuscate the memory construction. Since there are several aspects as they are mentioned in *Knowledge of System*, the *Expertise* is rated at *multiple experts*.

Knowledge of system: Some of the information or documentation of the SoCs is sometimes declared to be non-disclosable to the public (confidential). This implies a higher effort in the re-engineering. Furthermore, since the obfuscation is added to the system, particular approaches to de-obfuscate have to be identified. Either the attacker scans the whole main memory for a page they are looking for or they apply a statistical analysis to the permutation procedure. The former approach makes it necessary to assume that the attacker can scan the whole main memory. Furthermore, they need an evaluation function that determines whether or not the current scanned page is the one they were looking for. In other security fields such as cryptography, the strength of a certain function, such as encryption, is hard to define using discrete methods. Statistical analysis or complexity estimations on the randomization output forms the second approach to de-obfuscate the mapping table. This mathematical problem is comparable to the crypto-analysis of ciphertext. Concepts such as *known-ciphertext* and *chosen-plaintext* attacks might reveal algebraic weaknesses of the implemented algorithms. The proposed concept implies an in-depth target-specific reconnaissance to de-obfuscate the system mappings. Since the mappings are randomised on each system individually and change over time, given exploits are not directly applicable to a range of systems.

Window of opportunity: This is considered to be unlimited due to the fact that it was assumed that the attacker already found its way into the system. This metric remains unchanged.

Equipment: Handling embedded devices is subject to the application of bespoke equipment. This includes programming interfaces such as JTAG debuggers. This metric remains unchanged.

Residual Risk Definition

This section intends to revisit the risk analysis based on the case study of the Driver Information System which is introduced in Paragraph 3. Based on the risk of the hypothesised attack (**Hyp-Attack2**: *PE*'s memory base is tampered with by adjacent *PE*), the residual risk is evaluated involving the findings of the modified attack potential. Furthermore, the environmental aspects of the security requirements are briefly evaluated in the following section.

Residual Risk: Driver Information System. Considering the AP of 27 (*Beyond High*), the resulting AL on a low level. The effort to break the vulnerability is high, and therefore, the probability that an adversary takes this path is considerably small. Taking the *Impact Severity* from the case study into account, the resulting residual risk level is 2. The risk level is, therefore, lowered by two levels from a medium (4) down to low (2).

In the given setup, the low risk can be treated as acceptable. Here, the same conclusion applies referring to the evaluation of the domain-block mapping countermeasure. Higher impact severities lead to higher risks, and therefore a mitigation risk treatment strategy must be chosen to lower the risk.

Table 6.12 Residual risk analysis results.

Threat	Attack	AP	AL	Impact Severity	Residual Risk
Tampering	<i>PE</i> 's memory base is tampered with by adjacent <i>PE</i>	27	1	C2 S3	2

6.3 Comparison to Hypervisor-based System Architectures

This section aims at providing a brief comparison of the shown concepts to a typical Type-1 based hypervisor architecture. For that purpose, the DFD of the considered system is shown in Figure 6.3. The DFD models the domain separation paradigm as it is shown in 2.1.2.

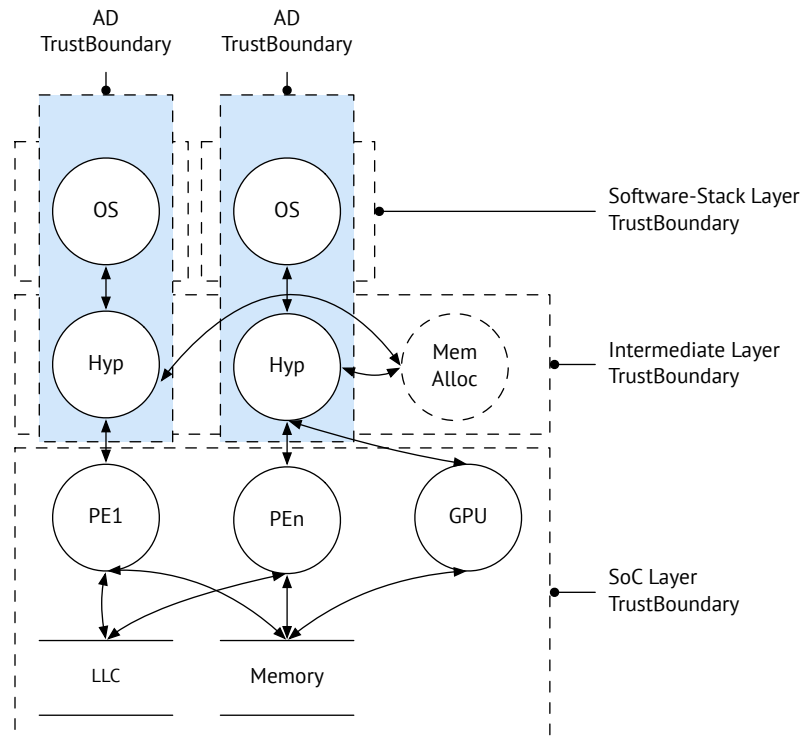


Fig. 6.3 DFD of the Type-1 hypervisor system architecture.

The general difference, between the AMP ToE and the Type-1 based system is the software instance on the intermediate layer. Furthermore, the trust boundaries from the AD and the intermediate layer overlap for the hypervisor processes (*Hyp*). This means that the separation relies on the capabilities of these layer entities. A general weakness of a hypervisor solution is that the software of the hypervisor instance is threatened to be compromised from lower privileged software. This could happen due to a malformed interface or erroneous implementations of the hypervisor facilities. In such a case, the compromised hypervisor process is capable of propagating through the system on behalf of an attacker. Compared to an AMP system, the hypervisor solution is prone to such hypothetical problems. In AMP systems, all intermediate layer interfaces are enforced in hardware which is less prone to modifications.

6.3.1 Attack Potential: Cache-Thrashing

The cache-thrashing pattern as it is conceptualized by this work will be considered and the resulting AP evaluated. For this consideration, it is important to define how the hypervisor processes map the memory between the address space of the software-

stack and the physical memory. There are two reasonable approaches: either in the hypervisor, a memory allocator (*Mem Alloc*) exists which is mapping the memory to the OSs on request or it is assumed that the hypervisor processes map the IPA to PA in the same way that the AMP does it. This means that, there is an identity mapping between the OS IPA addresses and the real physical addresses. In the latter case, the resulting AP would be similar to the compared AMP system, since the vulnerability is caused by the fixed identity mapping.

However, if the hypervisor implements a memory allocation mechanism the memory map might not be fixed identically or even deterministically. In that case, it is harder for the attacker to find the suitable PA in this partition to attack a specific WS. The resulting AP for that case is shown in Table 6.13 and discussed afterwards.

Table 6.13 Hypothesised attack: *OSquest* disrupts adjacent *OSquest's* access to *LLC*.

Factor	Level	Value
Elapsed time	≤ 3 Months	10
Expertise	Expert	6
Knowledge of system	Restricted	3
Window of opportunity	Unnecessary/unlimited	0
Equipment	Bespoke	7
AP: Beyond high		26

Rationale In the case when a non-deterministic memory allocation is using applying the cache-thrashing, the AP higher compared to the AMP variant. As a result, the likelihood of an attack is lower.

Elapsed time: The investigation and reconnaissance are expected to take approximately three months. Following the levels estimated for the other factors, it will take effort to gain knowledge of the internals of the caching infrastructure.

Expertise: A potential attacker needs to apply expert knowledge to mount the attack. MPSoCs or generally the embedded area deals with highly customized non-standardized architectures. Furthermore, a successful attack depend on the implementation of the memory allocation of the hypervisor. The re-engineering of concepts and designs is subject to rigid preparation since multiple system levels are involved. In this case, it is the SoC, intermediate and software-stack.

Knowledge of system: With a countermeasure applied, the attacker needs to investigate and re-engineer on a deeper level of detail. Notably, that not all information is publicly available.

Window of opportunity: This is considered to be unlimited since it was assumed that the attacker already found its way into the system.

Equipment: DoS is expected to be caused in a state where the device operates normally. The effects of an attack might be observable specialised measuring software, which is needed by the attacker to verify a successful attack.

6.3.2 Attack Potential: Tamper with Memory of Adjacent OS Guest

In this consideration, it is elaborated upon as to how the hypervisor solution relates to the AMP system with regards to breaching the memory protection. Similar to the previous consideration, the way in which the hypervisor implements the memory mapping is key to this question. Assuming a memory allocator with non-predictable memory mappings, it is generally more difficult for the attacker to determine its target memory area.

Table 6.14 Hypothesised attack: *OSguest* is tampered with by adjacent *OSguest*.

Factor	Level	Value
Elapsed time	≤ 3 Months	10
Expertise	Expert	6
Knowledge of system	Restricted	3
Window of opportunity	Unnecessary/unlimited	0
Equipment	Specialized	4
AP: High		23

Rationale

Elapsed time: The investigation and reconnaissance are expected to take approximately three months. Following the levels estimated for the other factors, it will take effort to gain knowledge of the internals of the memory allocation infrastructure.

Expertise: A potential attacker needs to apply expert knowledge to mount the attack. MPSoCs or generally the embedded area deals with highly customized non standardized architectures. The re-engineering of concepts and designs is subject to rigid preparation. This remains unchanged.

Knowledge of system: With a countermeasure applied, the attacker needs to investigate and re-engineer on a deeper level of detail. Notably, that not all information for the hardware and software is publicly available. This is justified by the fact that the industrial hypervisor derivations do not share the documentation with the public domain. However, there are open-source derivations, which share the documentation.

Window of opportunity: This is considered to be unlimited since it was assumed that the attacker already found its way into the system. This remains unchanged.

Equipment: DoS is expected to be caused in a state where the device operates normally. The effects of an attack might be observable specialised measuring software. Since some of the details are not considered to be documented, it is considered that the adversary needs to conduct specialized equipment to reveal the internals.

Potential Security Solutions

In a hypervisor driven environment, the feasibility to implement primary and secondary countermeasures is potentially higher. In a software-driven solution, including a certain infrastructure (for example IPC, memory allocation, scheduling, etc.), the realization of access controls, interception mechanisms and also obfuscation mechanisms become handier. However, the introduction of security mechanisms always comes along with new assets that need to be considered afterwards. Implementing protection mechanisms in software is considered to be prone to errors and must be assured in a rigid process [5].

6.4 Summary

This chapter described the results of the security evaluation process that have been applied to the proposed countermeasures. Furthermore, it briefly compared the results with a hypervisor-based system architecture.

For both countermeasures, a measurable decrease of the CVSS score as well as the corresponding AP was identified. In the case of the countermeasure introducing the domain-block mapping, the CVSS base score was reduced from *Medium* (5.3) to *Low* (1.9). The evaluation of the PoC implementation revealed that the memory mapping reduces the impact from ~ 133 cycles (arith. mean) down to ~ 9 cycles (arith. mean). However, there is still a raise compared to the normal duty measurements without the denial-of-service attack. In addition, the residual risk with regards to the driver information system context was lowered from risk level 4 down to low (1). The environmental score consideration did not affect the initial CVSS.

Referring to the memory-map shuffling the Base Score of the CVSS was reduced from 8.1 (High) down to 5 (Medium). The residual risk with regard to the case study was lowered from medium (4) down to low (2).

The comparison to a Type-1 hypervisor system architecture has shown that the AP and therefore the resulting attack likelihoods are higher, or lower respectively. Attackers would have to spend more effort to mount comparable attacks.

Related Work

If I have seen further, it is by standing on the shoulders of giants.

Sir Isaac Newton

Contents

7.1	Security Requirements Engineering	196
7.2	Security Architectures of AMP-based Systems	198
7.3	Offensive Methods and Attacks	198
7.4	Exploitation Prevention	200
7.5	Summary	202

This chapter collects and aligns with research in relation with the key contributions of this work. Since the contributions touch many different areas, here the content is structured in the respective disciplines.

7.1 Security Requirements Engineering

In [52], Glas et al. elaborate on the importance of the integration of safety and security requirements processes.

"A security process needs to be 'Safety Aware' and vice-a-versa." [52, p. 15]

They envisage the application of mechanisms, building blocks and patterns to achieve safety and security goals. Nonetheless, safety is considered in the impact severity category within this work. However, the effects on the safety integrity level (ASIL) is out of scope and, therefore, has not been considered.

Haley et al. present in [60] a framework for security requirement elicitation and analysis. The authors claim that adequate security requirements must satisfy three criteria: *definition*, *assumptions* and *satisfaction*. An important aspect in their work is:

"The system context is described using a problem-oriented notation, then is validated against the security requirements through construction of a satisfaction argument." [60, p. 1]

It is proposed to use an informal inner and a formal outer argument. Whereas the latter aims to formalize premises and behaviour in a logical representation, the former, which is the inner argument, seeks to give grounds and warrants to the premises of the outer argument. This complies to the method proposed in the CC security evaluation methodology [176].

In contrast to a formal definition and evaluation of security requirements, there are efforts to define security requirements which specify criteria that are evaluated. For example, Firesmith et al. specify in [46] a process for identifying highly reusable security requirements. The authors envisage an increase of the quality of systems by providing requirement templates. These templates refer to a quality model, which represents a hierarchical taxonomy of so-called quality sub-factors. A requirement template itself contains information on the asset, threat, attacker type and situation. In accordance with the security assessment process in this work, a target risk level is assigned to the requirement. In this way, it can be defined and qualified what the

predetermined acceptable risk level is. However, although the authors propose to apply a cost-benefit analysis, it is left unclear as to how the quality measure is reusable for other systems. In reference to this work, the differentiation between context-specific and independent quantifications of security aspects is key to define reusable security requirements.

In [38, 39] Elahi et al. conducted a survey on methods and frameworks for security requirements engineering. They mention three types of frameworks which include:

Agent and goal-oriented requirements frameworks that model and analyse security requirements based on the concept of trust, ownership, permission, and delegation. These concepts are dealt within the normal functional requirements model [51, 112].

Trust-based requirements frameworks that assert the relationship between entities to formalize and map them into the system requirements for implementation [184].

Risk and threat-based requirements frameworks that analyse threats and unwanted incidents for deriving security requirements [115, 116].

Elahi et al. furthermore give the Unified Modeling Language (UML) based on requirement frameworks in [38]. This is, however, considered to be the method for the notation of security requirements and therefore disregarded as a requirements framework. This work is conducted on the basis of a risk and threat-based requirements approach which is motivated by the fact that the risk and its components are used as qualification criteria to quantify the threats and countermeasures that are proposed throughout this work. In addition, risk and threat-based frameworks are adopted in automotive best practices [67, 81].

Related Protection Profiles

Separation kernels are widely accepted and discussed when it comes to the reliable and secure isolation of software stacks [107, 140]. Therefore, separation kernels are suitable to serve for an appropriate security problem definition. The EURO-MILS PP is taken into account in order to define the security problem definition [50, 128].

7.2 Security Architectures of AMP-based Systems

Numerous approaches exist for enforcing the separation of multiple logical domains running on MPSoCs. With respect to the architectural system layers, the proposals include solutions such as hypervisors, enhancements of the communication architecture and dedicated hardware elements.

Software-based solutions include hypervisor approaches such as micro-kernels [64, 183], type-2 hypervisors such as [33, 77] and commercial solutions such as [3, 69, 157, 185]. Nonetheless, the mentioned approaches are sound, valid and extensively used in the industry, and they are applied to the software-stack level of a system. Hence, they do not apply to the original goal of this work to propose a solution on the intermediate level.

Further research regards the CA of a MPSoC. Within the domain of NoC approaches address the separation of logical systems as well. For example, Porquet et al. introduces in [134] a lightweight architecture for embedded systems that allows multiple protection domains (compartments) to share processing, memory and other system resources securely. Furthermore, Inoue et al. show with VIRTUS: a new processor virtualisation architecture for security-oriented next-generation mobile terminals in [80]. Further examples found in [55, 141]. NoCs are an integral part of a MPSoC since they connect all system elements to each other. Hence, it is a central point to enforce protection policies.

Furthermore, some contributions focus on the introduction of additional hardware components to a MPSoC system. For example, Tan et al. [166, 165] propose an isolation unit that checks and maintains permissions at runtime.

7.3 Offensive Methods and Attacks

Automotive Security Attack Research

In the public domain as well as in the research community, automotive security turns increasingly into focus. As a result, potential adversaries will be able to attack remotely without physical access to the car. The attack surface in the past years was considered to be limited by the physical boundary of the vehicle. Plenty of researchers analysed the automotive E/E architecture such as the vehicular networks and ECUs [102]. As one of the first, Checkoway et al. analysed the attack surface of modern vehicles comprehensively [25]. The authors argue that,

"[...] the associated threat model — requiring prior physical access — has justifiably been viewed as unrealistic." [25, p. 1]

Consequently, the authors focus explicitly on externally accessible interfaces of the vehicle to show an until then non-recognized attack surface. For example, Miller et al. conducted a survey on automotive attack surface [121].

The research of Miller et al. [59, 122, 158] demonstrating a full breach of a *Chrysler Jeep Cheerokey* completely from a remotely connected personal computer. This has been the first time researchers were able to demonstrate a remote hack on a recently available vehicle. The breach forced the carmaker to recall about 1.4 million vehicles to fix the vulnerability [58]. As an effect of this, manufacturers increasingly became aware that the development of vehicular software needs to be done with a sophisticated security background.

Misuse of Shared Caches

The exploitation of shared LLC aiming for DoS attacks is rarely discussed in research. This is motivated by one reason. AMP-based systems or even more virtualized systems are not widespread in the automotive sector. So, there wasn't a necessity to investigate in such a direction, because technically the problem was solved by the application or operating system level. Nevertheless, there is research dealing with the mechanics of locating and filling designated way-sets. Osvik et al. discuss and formalises in [129] two methods to mount cross-core measurements based on LLC: *Prime+Probe* and *Evict+Time*. Essentially, the methods include an algorithm to fill a specific way-set in LLC, which is either the *Prime* or *Evict* part of the methods. The vast majority of the work deals with timing attacks on Advanced Encryption Standard (AES)

DMA based Attacks

Vulnerabilities and attack vectors on embedded multi-operating systems are rarely available. Research on direct mapped devices comes close to the topic discussed in this document. As an example, Lone Sang et al. [142] and Boileau [16] show exploitations of vulnerabilities for DMA capable devices. Furthermore, this concept is also adopted by other researchers. Danisevskis et al., for example show in their work a similar approach which was conducted using a smartphone SoC [35].

7.4 Exploitation Prevention

An appropriate solution to prevent the intrusion is to build an authentication mechanism into the communication over the *CSB*. This aims at identifying and passing only permitted accesses to memory. As an example, Porquet et al. proposes in [135] an approach to enforce system isolation based on the extensions in the hardware architecture. Furthermore, their advantages are introduced in NoC-MPE: a secure architecture for flexible co-hosting on shared memory MPSoCs, and from Fiorin et al. [45], who show secure memory accesses on NoC. Furthermore, in [29] Coburn et al. demonstrate with *SECA* a security-enhanced communication architecture to enforce system isolation during data transfer in systems utilizing *AMBA*-Protocols for communication.

AMD [6], Intel [71] and ARM [120] introduced techniques to protect virtualized environments against bus mastering devices that might be applicable in the *AMP* based context. Nevertheless, these mechanisms need an adaptation of the hardware layout and a common basis for the virtual address mappings.

Cache-Thrashing Mitigation

Concerning the issue of cache thrashing or contention, it is subject to a plethora of research contributions. In the following, the key contributions are analysed. The contributions are analysed concerning the approach to solve cache thrashing in general, the area in which the approach is applied and the specific implementation.

Particularly, on software-stack (OS) has a high number of applications competing for space in the cache. Accordingly, solutions to manage the cache allocation indirectly are approached by many contributions [27, 94, 164]. Mostly, the applied method to manage the cache is page colouring [113]. Furthermore, concepts and approaches to manage the page colours during runtime are proposed [28, 92, 130, 136, 163, 191]. The solutions are implemented into the memory allocation mechanism on OS level. As an example, Ying et al. [189] encounter the issue of cache overcommitment caused by dynamically occurring threats.

"(...) presents a memory management framework called COLORIS, which provides support for both static and dynamic cache partitioning using page colouring." [189, p. 1]

The intention in applying cache colouring is mostly to achieve real-time computing goals, in order to increase the determinism of memory accesses.

Approaching the issue of cache thrashing in virtualized environments is conducted by Chen et al. [89] and in Tam et al. [164]. The approaches aim to implement the page colouring into a VMM or hypervisor.

Within the embedded research, Valsan et al. [181] analyse the issue of cache contention of MSRHR registers in the caches of COTS processors. The authors claim that the isolation using page colouring techniques is not effective for real-time use cases. Hence, a collaborative hardware and software approach has been proposed and evaluated using a cycle accurate system simulator. On the contrary to previous contributions, the authors analyse effects of shared hardware isolation beyond way-sets.

The state-of-the-art techniques reviewed here, show a wide field of solutions in cache management. Most prominently, the approach of page/cache colouring is facilitated in complex appliances. Accordingly, the concept of page colouring is a reasonable and discussed approach for encountering shared cache contention. However, the approaches are applied on SMP-based systems. Even the proposals aiming at VMM/hypervisor level do not consider the characteristics of AMP-based systems.

Memory-Map Shuffling

The idea of obfuscating addresses is not new in certain areas. On an application level, address obfuscation is adopted by many operating systems. Particularly, in general purpose operating systems such as *Linux*, *Windows* and *Mac OSX* this technique is actually state-of-the art. As of today, this is said to be one of the most effective countermeasures against memory exploits. Recent efforts brought that technique down to the system level, by randomizing the operating system's kernel address space [37]. In Linux, for example, the developers aimed for a significant increase in system security by making attacks into the monolithic kernel space less predictable for adversaries.

In [114], Gispert et al. analyse the effectiveness of 64bit ASLR in a Linux operating system. Amongst a proof-of-concept attack to dramatically reduce the entropy of the obfuscation method in the particular implementation, they introduce tree dimensions of the effectiveness of ASLR. These includes: all areas of the memory layout must be randomized to defeat attackers: the range of entropy must be as high as possible, the relocation frequency is important to determine how much time an adversary has to break a particular randomized memory layout. These aspects are taken into account in the herein proposed memory-shuffling.

In [13], Bhatkar et al. describe address obfuscation as an efficient approach for combating memory error exploits. The authors argue that these attacks require an attacker to have an in-depth understanding of the internal details of a victim program,

including the locations of critical data and/or code. Therefore, program obfuscation is a general technique for securing programs by making it difficult for attackers to acquire such a detailed understanding. Kil et al. extend on [93] the idea of address space layout permutation to enable a finer-grained randomization. Generally, they address one of the biggest drawbacks of current address space randomization implementations, namely the lack of a cryptographic secure entropy. Possible adversaries are able to guess the locations statistically in a very short amount of time since the number of bits used for randomization is very limited.

The permutation of address layouts facilitate to combat attacks using techniques such as buffer-overflows, format string attacks and code re-usage attacks like ROP. In [178], Shuo et al. introduces a method to utilize hardware virtualization in order to prevent ROP attacks within the kernel. In [139], Rushanan et al. elaborate on the concept of malicious behavior based on DMA transfers. The attacks are implemented using commodity desktop hardware. Although the implementation is not applicable to embedded hardware, the DMA issue is transferable to the attack surface of embedded SoC.

7.5 Summary

This chapter has shown the content and relationship to research the areas of security requirements and offensive and defensive approaches. The vast majority of research focuses on solutions to hypothetical treat and attack scenarios. These solutions are presented as architectural improvements and mostly evaluated by conceptual analysis and laboratory experiments. A crucial missing aspect is the structural evaluation of security requirements and commonly accepted quantification models. In other words, it is answered as to how feasible it is to implement the proposed solutions, rather than to give a statement on its security impact.

Contents

8.1	Contents	204
8.2	Contributions of this Research	206
8.3	Answers to the Research Questions	208
8.3.1	Research Question 1	208
8.3.2	Research Question 2	209
8.4	Limitations	209
8.5	Future Work	210
8.6	Summary	211

"There are more things in heaven and earth, Horatio, than are dreamt of in your philosophy."

William Shakespeare

This chapter concludes on the results of this work. Concerning the research questions, the key contributions are elaborated upon. Furthermore, its limitations are described. The chapter concludes with an overview of future work that is based on the findings of this research.

8.1 Contents

Fundamental Concepts. An introduction of fundamental concepts profiled AMP based systems. Furthermore, a security assessment and evaluation process has been introduced. This chapter concludes with a description of the research methodology. The following research questions are formulated due to the previous problem analysis and foundation review. Furthermore, the research artefacts and methods are detailed. Primarily, developmental methods are applied, including formulative and descriptive methods. Accordingly, the research artefacts are descriptive and formulative models. Case studies are conducted to show the applicability concerning two cases. The evaluation of the resulting proof-of-concept implementation is conducted by observing laboratory experiments.

Risk Assessment of a Driver Information System. The driver information system implements an example throughout the thesis. The case study realizes the approach of a mixed-criticality system in an automotive deployment. Functionally, it consolidates two typical automotive functional domains. It includes a cluster instrument domain, which operates a common display and the HMI to the driver. Furthermore, there is also an infotainment domain which mainly operates media and connectivity functions. These two domains run a common MPSoC Platform which facilitates the AMP system utilization paradigm.

Furthermore, a comprehensive risk assessment of the previously described driver information system is conducted. Threat and attack analysis input a set of attacks to the assessment. It is considered that the motivation of an adversary is to control one of the functional domains within the ToE. Furthermore, the disruption of a service has been anticipated. The result of the assessment revealed two hypothesised attacks that will be assessed in further detail in the following chapters.

Vulnerability Assessment. The vulnerability assessment conceptually analysed the vulnerability and exploitation of the previously hypothesised attacks. First, the disruption of the accesses to a shared last level cache was examined. Second, the assessment on the tampering of memory by an adjacent processing element has been conducted.

It has been shown a denial-of-service attack on shared cache setups of modern processing element designs. The contention is caused by shared way-sets in k-way-set associative caches. The concept shows how to provoke the contention by overcommitting associated way-sets. As a result, the vulnerability is a non-controlled shared usage of cache way-sets which enables an adversary to provoke cache-misses on memory accesses.

The exploitation of the identified vulnerability has been conceptually and experimentally approached. The assessment reveals that particular physical addresses can be modified and cause the previously described effects. This is possible even though both systems operate on separated memory partitions of asynchronous domains in an AMP system. In a penetration test, the formerly conceptualized attack was experimentally demonstrated.

The second assessment approached concepts of misusing a memory access capability in heterogeneous MPSoC designs. DMA capable peripherals such as GPUs and co-processors can be misused to exploit an insufficient protection architecture within the MPSoC. The conceptual exploitation has been analysed.

Experimentally, the exploitation of the identified vulnerability has been explored and scored. The penetration test demonstrated a compromise of a co-processor firmware image to circumvent the memory protection of the main processors.

Both assessments were reflected in the third part. A general view of the given vulnerabilities and assets is conducted. For that purpose, the concepts have been generalized and reflected in the layer hierarchy of an AMP system. The results are twofold. First, the systems should be analysed reflecting the distinct hierarchies of an AMP system. This will reveal resource control and access violations. Furthermore, the findings enable the defining of requirements for protection architectures such as multi-layered systems. As a result, the security problem statement of the envisioned security pattern have been analysed.

Risk Treatment. The treatment of risks that were identified in the previous work is elaborated upon. It, therefore, completes the consideration of security patterns defined throughout this work. For that purpose, here, exploitation prevention mechanisms and

a denial-of-service mitigation have been proposed. Based on the generalized findings of the vulnerability assessment, the concept of differentiating countermeasures in primary and secondary has been proposed. A primary countermeasure defeats a particular vulnerability, whereas a secondary countermeasure aims at preventing from unknown vulnerabilities or exploitation attempts. Along with the following countermeasures, this chapter completes the security patterns with these security solutions.

Taking these concepts into account, a domain-block memory-mapping has been introduced. This concept realizes an effective mitigation against denial-of-service attacks against shared way-sets in a LLC. It realizes a primary countermeasure. By applying the proposed mapping, ADs work on private memory-chunks (domain-blocks) in the shared cache. An enforced eviction of cache-lines by an adversary is prevented with this technique. A limitation of the approach is that the cache will be statically divided between the ADs. For applications seeking for the best-effort performance, this might have an impact.

Furthermore, in this chapter, a concept to mitigate the effects of breaching the hardware memory protection is proposed. Here, a secondary countermeasure preventing the exploitation has been designed. The obfuscation of the intermediate address mapping is based on the introduction of random permutations of a normal, continuous memory page arrangement. The so-called "memory-map shuffling" increases the effort to reinterpret the memory structures. The shuffled-memory map is transparent to higher layers in the system architecture. Revisiting the system properties of the experimental platform, the proposed approach reaches 20 bits of entropy, which is comparable to common ASLR implementations of 32-bit system architectures.

Security Evaluation The results of the security evaluation process that has been applied to the proposed countermeasures. Furthermore, it briefly compared the results with a hypervisor-based system architecture.

8.2 Contributions of this Research

Contributions are made concerning the research questions motivated in Section 2.3. Hence, the main findings are structured into the two main questions of this work. Also, results that surround the contributions as mentioned earlier are described.

Overview of the main contributions:

Security context profile of AMP-based systems: The profile defines all important aspects of an AMP system in order to conduct a rigorous security

assessment. For that purpose, it differentiates the logical aspects of such a class of system. Furthermore, the key elements of the considered hardware architecture are described. The profile makes the considered system analysable.

Conduction of a security assessment process to create and evaluate novel security patterns. The process combines several security engineering techniques, methods and tools in order to define security patterns for the intended system. It takes hypothetical risk analysis methods into account and combines it with technical vulnerability assessment methods. The qualification and quantification are based on well-accepted metrics and provides by this token comparability and transferability of results.

Conduction of a case study to apply the security context profile and assessment process in an automotive scenario. This case study shows how the context profile can be realized in an automotive scenario. Furthermore, the application of mixed-criticality of asynchronous domains of an AMP system is shown.

Assessment of vulnerabilities and their conceptual and practical exploitation. Here, the conceptual aspects of a vulnerability in the utilization of shared k-way-set caches are assessed in detail. Furthermore, the tampering and elevation of privilege by memory protection breaches are conceptualized. On that basis, the conceptual exploitation is shown and derives the context-specific exploitation factors. These factors are practically evaluated on the experimental platform by conducting a penetration test. Both results contribute to the security of future systems by providing a rigorous technical background as well as practical evidence that those kinds of vulnerabilities are exploitable in real system setups.

Derivation of a generalized security problem pattern. Whereas, the assessment of particular vulnerability instances shows a single problem, the generalized vulnerability pattern presents a broader view. Here, it is shown how the security problems evolve. This is important to state a holistic security pattern.

Derivation of risk treatment strategies for the AMP specific intermediate level. Here, the concept of primary and secondary security countermeasures is introduced. Using that definition, the necessity of a secondary countermeasure on the intermediate level has been revealed. The strategy to use the memory-map for both countermeasure types is conceptualized.

Conceptual analysis and security evaluation of a novel risk mitigation mechanisms. The two major vulnerabilities are mitigated due to this contribution. For the denial-of-service attack to the cache, the well-known concept of cache colouring has been adopted and applied to create a memory-mapping that prevents a shared usage of cache-way sets. For the second attack type, considering the breach of memory protection mechanisms, the concept of obfuscation has been adopted, conceptually analysed and applied to the context profile. Using this method, the lack of secondary countermeasures on the intermediate level can be solved.

8.3 Answers to the Research Questions

8.3.1 Research Question 1

Research Question 1: *What is the pattern of DMA and cache-thrashing attacks reflecting the particular AMP characteristics?*

In the attack pattern, a malicious asynchronous domain seeks to slow down the other target domain to cause availability issues. Through this, the concept shows how to deliberately disrupt the memory access latency to a particular memory address. The analysis indicates that due to the association of physical memory and way-sets within the LLC, a so-called cache-thrashing effect is possible. This applies even if the memory partitions are separated using appropriate hardware mechanisms such as the MMU. The given approach might impact the availability and therefore the WCET of applications running in the target domain. As a result, the vulnerability is the static association of cache way-sets to the physical memory in conjunction with a continuous memory mapping. The exploitation is fairly easy because it is executable just by accessing private memory addresses. However, an important preliminary is that the adversary knows the caching parameters as well as the exact physical address of its target. The penetration test has shown that the memory access of a victim system can significantly force a higher latency.

AMP-based systems foster the problem of inappropriate or erroneous hardware design. In this work, it has been observed that this issue is twofold. First, since the AMP-paradigm is quite new in MPSoC appliances, the hardware design suffers from inconsistent protection architectures. This can be shown by the possibility of DMA-based attacks that are exploitable due to the lack of distributed access control

mechanisms to each DMA-capable hardware element. The non-existence of a software-based access control mechanism, for example in a hypervisor, emphasizes the fact that total separation is not facilitated by all system layers. The second aspect is design errors that weren't anticipated during the development of the hardware. It could always be the case that some of the protection mechanisms fail or can be circumvented due to new techniques or simply due to errors in the implementation. As a result, the vulnerability is an erroneous security protection architecture. The exploitation relies on the low-level usage of hardware capabilities and reconnaissance of the system internals.

8.3.2 Research Question 2

Research Question 2: *How can one mitigate the risk of exploitation of the previously identified vulnerabilities?*

In general, by the utilization of memory-maps, the concepts of primary and secondary countermeasures are realizable. This is particularly true for the two countermeasures proposed in this work.

The proposed security solution structures the system memory-map in a way to avoid interference with concurrent asynchronous domains in the ToE. In this case, a primary countermeasure is realized. It instantiates the risk treatment of the denial-of-service attack against the shared way-sets in a cache. The surface of interference is removed by the use of resource separation.

Furthermore, the memory-map shuffling mitigates the risk of exploitation regarding the memory access control breach. The concept proposes the obfuscation of the system memory-map to increase the effort of memory-based attacks. This implements the concept of a secondary countermeasure. Due to the precondition that the primary countermeasures have already failed, the existence of an exploitation prevention mechanisms on the intermediate level is proposed. In a decent defence-in-depth security architecture this adds another hurdle for potential adversaries.

8.4 Limitations

There is no total freedom of interference between the two domains, which is demonstrated by the still measurable impact even with the applied domain-blocks applied. Here, one of the limitations of the approach becomes visible. The mapping only controls the separation of way-sets in a cache. If the cache implements shared interfaces to

transfer data between the cache levels, there is no possibility to control the competing accesses with the proposed approach. Hence, this surface for interference depends on the particular hardware implementation and has to be considered. A reasonable consideration would be the WCET requirements of applications ran by the target. If the worst-case is still acceptable, which means including the maximum impact, then the solution is applicable. As a result, the measurements shows the worst-case scenario.

Furthermore, the introduction of the memory-map shuffle introduces a new asset which is the translation table. Although, this doesn't contradict well-accepted security principles, such as Kerkhoffs principle¹, the protection of the translation table must be taken into account during the design of a system.

8.5 Future Work

Among the contributions in this thesis, further directions for research have been identified.

In this work, a single and particular layer was focussed. In future work, the question could be raised on how: is it possible to define a rule-set which applies to all system layers of any kind, which enables rigid isolation of logical entities? This rule is than usable to check whether or not the architecture is vulnerable or in turn it could be used as a guideline for the architectural design.

This work relied on well-accepted quantification methods to evaluate vulnerabilities, exploitation and the effectiveness of countermeasures. The CVSS meaningfully categorizes aspects of vulnerabilities and their exploitation. However, for a detailed differentiation, the scales are sometimes too coarse. In order to incorporate this scoring system in formalized security patterns a fine-grained scale for all aspects would sharpen the impacts of certain properties of vulnerability. For example, in this work, so-called exploitability factors have been introduced. The intention of these is to express the exact impact of a particular attack. Therefore, context-dependent score factors that can be incorporated in the CVSS are proposed for future work.

The security of a system is continuous and always changing target. It is always about the identification and reaction to problems. Static system components often make an adaption to new situations impossible. Therefore, future system designs

¹A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

should take this into account and put more effort into the full *adaptivity* of systems. Even on the hardware layer, dynamic hardware elements, which can re-composite, for example, the MPSoC architecture, would enable more degrees of freedom to realize countermeasures.

The proposed memory-map shuffling concept could be transferred to higher system levels. For example, the MMUs utilized to manage the process space of applications are suitable examples. This could be an alternative to the wide-spread ASLR mechanisms, while having the advantage of totally obfuscating the memory structure, rather than only adapting the relative distances. However, it needs to be investigated how linked libraries, shared memory and other facilities relate to this concept. Furthermore, in the proposed concept the Fisher-Yates algorithm has been adopted to shuffle the memory-map entries. There are numerous other algorithms that are applicable. The here used algorithm is handy and straight-forward in order to apply it in a PoC. However, other algorithms might be even more efficient for the intended purpose.

The PoC measurements of the domain-block concept revealed that there is still a certain amount of interference when two domains are competing for a particular way-set in the cache. The reasons for that could be subject to further investigation. Here, the competing access to the interface which transfers the data from cache level to cache level is a reasonable source for the measured impact. Diving deeper into the architectural internals potentially reveal further attack surfaces or could increase the effectiveness of the proposed concept to limit the impact of a DoS attack.

8.6 Summary

This chapter has concluded on the results of this thesis improving the security of AMP-based systems in an automotive context. This was achieved by introducing the concept of primary and secondary countermeasures by the utilisation of system memory-maps. Former static and predictable system configurations are now backed by secondary exploitation prevention mechanisms that apply when the primary protections fail, or new vulnerabilities are exploited. This has been achieved by the analysis of two known attack phenomena in the context of AMP-based systems. The findings of these attacks have been generalised and made applicable to a broad range of systems applying similar system paradigms. By the proposal and rigid application of the primary/secondary countermeasure approach, the concepts of *domain-block* memory-mapping for

cache-thrashing prevention and the *memory-map shuffling* have been proposed. The latter approach mitigates exploitations on the integrity of control flows and data within the memory. Moreover, the former prevents denial-of-service attacks on shared last level caches.

Facilitating systems employing the AMP-paradigm has its neat advantages from an implementation point of view. Amongst others, an important advantage is the minimal performance impact. However, by moving the separation capabilities into the SoC-layer security issues might arise. There is no further line of defence left to fill unknown future security gaps. Accordingly, the possibilities are limited to adapt to threats or vulnerabilities that have been observed after the deployment. Particularly, secondary countermeasures help to prevent the exploitation of compromised primary measures.

ADAS	Advanced Driver Assistant Systems
AD	Asynchronous Domain
AES	Advanced Encryption Standard
AL	Attack Likelihood
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multiprocessing
AP	Attack Potential
ASIL	Automotive Safety Integrity Levels
ASLR	Address Space Layout Randomization
AXI	Advanced eXtensible Interface
BCM	Body Control Unit
BSP	Board Support Package
CAN	Controller Area Network
CAPEC	Common Attack Pattern Enumeration and Classification
CoA	Conceptual Analysis
CC	Common Criteria

CE	Consumer Electronics
CFI	Control Flow Integrity
CI	Communication Infrastructure
CIA	Confidentiality Integrity Availability
CImpl	Concept Implementation
CL	Cache Line
COTS	Commercial Of The Shelf
CPU	Central Processing Unit
CS	Case Study
CVE	Common Vulnerabilities and Exposures
CVE	Common Vulnerability Evaluation
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
CA	Communication Architecture
CoPE	Companion Processing Element
DCU	Domain Control Unit
DDR	Double Data Rate
DFD	Data Flow Diagram
DI	Driver Information
DMA	Direct Memory Access
DSP	Digital Stream Processor
DoS	Denial of Service
E/E	Electric/Electronic
EAL	Evaluation Assurance Level

ECU	Electronic Control Unit
EoP	Elevation of Privilege
ELF	Executable and Linking Format
ESP	Electronic Stabilization Program
EVITA	E-Safety Vehicle Intrusion Protected Applications
EVM	Evaluation Module
FAIR	Factor Analysis of Information Risk
FPGA	Field Programmable Gate Array
GPOS	General Purpose Operating System
GPU	Graphical Processing Unit
HARA	Hazard and Risk Assessment
HEAVENS	HEAling Vulnerabilities to ENhance Software Security and Safety
HMI	Human Machine Interface
IPA	Intermediate Physical Address
IPC	Inter Process Communication
IPU	Image Processing Unit
IPL	Initial Program Loader
IP	Intellectual Property
IPMMU	Intellectual Property Memory Management Unit
IoT	Internet-of-Things
ISR	Interrupt Service Routine
IVI	In-Vehicle Infotainment
JTAG	Joint Test Action Group
KASLR	Kernel Address Space Layout Randomization

LE	Laboratory Experiment
LIN	Local Interconnect Network
LLC	Last Level Cache
LPAE	Large Physical Address Extension
LRU	Least Recently Used
LTE	Long Term Evolution
MA	Memory Architecture
MB	Memory Block
MC-system	Mixed Criticality System
MCS	Mixed-Criticality System
MC	Mixed-Criticality
MESI	Modified Exclusive Shared Invalid
MILS	Multiple Independent Levels of Security
MIMD	Multiple Instructions Multiple Data
ML	Memory Line
MMU	Memory Management Unit
MOST	Media Oriented Systems Transport
MPSoC	Multiprocessor System-on-Chip
MPU	Memory Protection Unit
MSHR	Miss Status Holding Registers
NIC	Network Interface Controller
NIST	National Institute of Standards and Technology
NUMA	Non Uniform Memory Architecture
NVD	National Vulnerability Database

NoC	Network-on-Chip
ODB	On Board Diagnostics
OEM	Original Equipment Manufacturer
OS	Operating System
OTA	Over-the-Air
PA	Physical Address
PE	Processing Element
PLC	Product Life Cycle
PP	Protection Profile
PMCCNTR	Performance Monitor Control Cycle Counter
PU	Processing Unit
PoC	Proof-of-Concept
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
ROM	Read Only Memory
ROP	Return Oriented Programming
RTOS	Real-time Operating System
SAR	Security Assurance Requirement
SDRAM	Synchronous Dynamic Random Access Memory
SFR	Security Functional Requirement
SHM	Shared Memory
SMP	Symmetric Multi Processing
SMMU	System Memory Management Unit

SPI	Serial Peripheral Interface
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, Elevation of Privilege
ST	Security Target
SoC	System-on-Chip
SOP	Start of Production
THROP	Threat and Operability Analysis
TB	Trust Boundary
TLB	Translation Lookaside Buffer
TSF	Target Security Function
ToE	Target of Evaluation
UMA	Unified Memory Architecture
UML	Unified Modeling Language
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VA	Virtual Address
VE	Virtualization Extension
VMM	Virtual Machine Monitor
VS	Vulnerability Score
WCET	Worst Case Execution Time
WiFi	Wireless Local Area Networking
WS	Way-Set
eCall	Emergency Call Interfaces
eMMC	Embedded Multi Media Card

mobileOS Mobile Operating System

multi-OS Multi Operating System

References

- [1] IEEE standard test access port and boundary-scan architecture. Technical report, 2001. URL <http://dx.doi.org/10.1109/ieeestd.2001.92950>.
- [2] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):4–40, October 2009.
- [3] SYSGO AG. PikeOS Hypervisor, 2017. URL <https://www.sysgo.com/products/pikeos-hypervisor/>. [Accessed January 5st, 2018].
- [4] J Alves-Foss, C Taylor, and P Oman. A multi-layered approach to security in high assurance systems. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, page 10 pp. IEEE, 2004.
- [5] J Alves-Foss, P W Oman, and C Taylor. The MILS architecture for high-assurance embedded systems. . . . *of embedded systems*, 2006.
- [6] I AMD. O virtualization technology (iommu) specification. *AMD Pub*, (34434), 2007.
- [7] R.J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2010. ISBN 9781118008362. URL http://books.google.de/books?id=eo4Otm_TcW8C.
- [8] Birnie Andy and Timo van Roermund. A Multi-Layer Vehicle Security Framework. pages 1–18, October 2016.
- [9] ARM. Cortex-A15 Processor. Technical report, ARM Limited, 2011.
- [10] ARM. ARM Architecture Reference Manual - ARMv7-A and ARMv7-R edition. 2012.

- [11] AMBA Arm. AXI Protocol Specification. Technical report, 2004.
- [12] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, Maurizio Peri, and Saverio Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pages 170–177, New York, NY, USA, 2003. ACM. ISBN 1-58113-676-5. doi: 10.1145/951710.951734. URL <http://doi.acm.org/10.1145/951710.951734>.
- [13] S Bhatkar, D C DuVarney, and R Sekar. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. *USENIX Security*, 2003.
- [14] H Bidgoli. *Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management*. Handbook of Information Security. Wiley, 2006.
- [15] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of Network-on-chip. *ACM Computing Surveys*, 38(1):1–es, June 2006.
- [16] Adam Boileau. Hit by a Bus:Physical Access Attacks with Firewire. pages 1–36, October 2006.
- [17] Adam Boileau. Hit by a bus: Physical access attacks with firewire. *Presentation, Ruxcon*, page 3, 2006. URL https://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf. [Accessed January 5st, 2018].
- [18] Anton Bretting and Mei Ha. Vehicle control unit security using open source autosar. 2015.
- [19] Manfred Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 33–42. ACM, 2006.
- [20] Peter David Bruza, Th P van der Weide, and Theodorus Petrus Weide. The Semantics of Data Flow Diagrams, 1989.
- [21] Alan Burns and Robert Davis. Mixed criticality systems - a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.
- [22] W. O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, Sungjoo Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava. Multiprocessor soc platforms: a component-based design approach. *IEEE Design Test of Computers*, 19(6):52–63, Nov 2002. ISSN 0740-7475. doi: 10.1109/MDT.2002.1047744.
- [23] Mark Chang. *Principles of Scientific Methods*. CRC Press, Taylor & Francis Group, LLC, CRC Press, Taylor & Francis Group, LLC, 2014.

- [24] Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, and Marcel Winandy. Return-oriented programming without returns. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 559–572. ACM, 2010.
- [25] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.
- [26] Shuo Chen, Jun Xu, Emre C Sezer, Prachi Gauriar, and Ravishankar K Iyer. Non-control-data attacks are realistic threats. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, pages 12–12. North Carolina State University, USENIX Association, July 2005.
- [27] Sangyeun Cho and Lei Jin. Managing distributed, shared l2 caches through os-level page allocation. In *IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE*, pages 455–468. IEEE Computer Society, 2006.
- [28] Sangyeun Cho, Lei Jin, and Kiyeon Lee. Achieving Predictable Performance with On-Chip Shared L2 Caches for Manycore-Based Real-Time Systems. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, pages 3–11. IEEE, 2007.
- [29] Joel Coburn, Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. SECA: security-enhanced communication architecture. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 78–89, New York, New York, USA, September 2005. ACM.
- [30] Walls Colin. Asymmetric Multi-Processing (AMP) vs. Symmetric Multi-Processing (SMP). Technical report. URL <http://scitechconnect.elsevier.com/asymmetric-multi-processing-amp-vs-symmetric-multi-processing-smp/>.
- [31] TIS Committee. Tool interface standard (tis) executable and linking format (elf) specification. pages 1–37, 1995.
- [32] NIST Computer Security Division CSD. NIST SP 800-126 Revision 2, The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1. pages 1–58, September 2011.
- [33] Christoffer Dall and Jason Nieh. Kvm/arm: the design and implementation of the linux arm hypervisor. In *Acm Sigplan Notices*, volume 49, pages 333–348. ACM, 2014.
- [34] Martin Horn (eds.) Daniel Watzenig. *Automated Driving: Safer and More Efficient Future Driving*. Springer International Publishing, 2017.

- [35] Janis Danisevskis, Marta Piekarska, and Jean-Pierre Seifert. Dark side of the shader: Mobile gpu-aided malware delivery. In *Information Security and Cryptology-ICISC 2013*, pages 483–495. Springer, 2014.
- [36] DENX. Das U-Boot – the Universal Boot Loader, 2013. URL <http://www.denx.de/wiki/U-Boot>. [Accessed January 5st, 2018].
- [37] Jake Edge. Kernel address space layout randomization. *Linux Security Summit*, October 2013. URL <http://lwn.net/Articles/569635/>.
- [38] Golnaz Elahi. Security requirements engineering: state of the art and practice and challenges. Technical report, 2009.
- [39] Golnaz Elahi, Eric Yu, Tong Li, and Lin Liu. Security Requirements Engineering in the Wild: A Survey of Common Practices. In *2011 IEEE 35th Annual Computer Software and Applications Conference - COMPSAC 2011*, pages 314–319. IEEE, 2011.
- [40] Jon Erickson. *Hacking: The art of exploitation*. No Starch Press, 2008.
- [41] Dieter Ernst, Sheri Martin, and East-West Center. The Common Criteria for Information Technology Security Evaluation, 2012.
- [42] ETSI. ETSI TS 102 165-1. pages 1–79, March 2011.
- [43] Germany Federal Office for Information Security. The PP/ST Guide August 2010 Version 2 Revision 0. pages 1–78, October 2010.
- [44] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011.
- [45] L Fiorin, G Palermo, S Lukovic, V Catalano, and C Silvano. Secure Memory Accesses on Networks-on-Chip. *Computers, IEEE Transactions on*, 57(9):1216–1229, 2008.
- [46] Donald Firesmith. Specifying Reusable Security Requirements. *Journal of Object Technology*, 3(1):61, 2004.
- [47] FIRST.Org, Inc. Common Vulnerability Scoring System v3.0. pages 1–15, July 2015.
- [48] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949.
- [49] Jack Freund and Jack Jones. *Measuring and Managing Information Risk: A FAIR Approach*. Butterworth-Heinemann, 2014.

- [50] Igor Furgel and Iola Saftig. 2015-EURO-MILS-Protection-Profile-White-Paper-V1.2. pages 1–66, March 2015.
- [51] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. *RE*, 2005.
- [52] Benjamin Glas, Carsten Gebauer, Jochen Hänger, Andreas Heyl, Jürgen Klarman, Stefan Kriso, Priyamvada Vembar, and Philipp Wörz. Automotive Safety and Security Integration Challenges. *Automotive - Safety & Security*, 2014.
- [53] Robert L Glass, Iris Vessey, and Venkataraman Ramesh. Research in software engineering: an analysis of the literature. *Information & Software Technology* (), 44(8):491–506, 2002.
- [54] Robert L Glass, V Ramesh, and Iris Vessey. An analysis of research in computing disciplines. *Communications of the ACM*, 47(6):89–94, June 2004.
- [55] Kees Goossens, Martijn Koedam, Andrew Nelson, Shubhendu Sinha, Sven Goossens, Yonghui Li, Gabriela Breaban, Reinier van Kampenhout, Rasool Tavakoli, Juan Valencia, et al. Noc-based multiprocessor architecture for mixed-time-criticality applications. *Handbook of Hardware/Software Codesign*, pages 491–530, 2017.
- [56] Mel Gorman. Understanding The Linux Virtual Memory Manager. pages 1–731, July 2007.
- [57] D. Goswami, M. Lukasiewicz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1227–1232, March 2012. doi: 10.1109/DATE.2012.6176680.
- [58] Andy Greenberg. After Jeep Hack, Chrysler Recalls 1.4M Vehicles for Bug Fix. July 2015.
- [59] Andy Greenberg. Hackers Remotely Kill a Jeep on the Highway—With Me in It, July 2015. URL <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>. [Accessed January 5st, 2018].
- [60] C B Haley, R Laney, J D Moffett, and B Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153.
- [61] Seth Hanford. Common Vulnerability Scoring System v3.0: Specification Document. pages 1–21, July 2015.

- [62] Allen Harper, Shon Harris, Jonathan Ness, Chris Eagle, Gideon Lenkey, and Terron Williams. *Gray Hat Hacking The Ethical Hackers Handbook*. McGraw-Hill Osborne Media, 2011.
- [63] Gernot Heiser. Hypervisors for consumer electronics. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5. IEEE, 2009.
- [64] Gernot Heiser and Ben Leslie. The okl4 microvisor: Convergence point of microkernels and hypervisors. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, pages 19–24. ACM, 2010.
- [65] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.
- [66] Olaf Henniger. E-Safety Vehicle Intrusion Protected Applications. pages 1–2, April 2011.
- [67] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Wey. *Security requirements for automotive on-board networks*. IEEE, 2009.
- [68] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the stride approach. <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx>, 2006.
- [69] Dan Hildebrand. An architectural overview of qnx. In *USENIX Workshop on Microkernels and Other Kernel Architectures*, pages 113–126, 1992.
- [70] Mark D Hill and Alan Jay Smith. Evaluating Associativity in CPU Caches. *IEEE Trans. Computers*, 1989.
- [71] R Hiremane. Intel virtualization technology for directed i/o (intel vt-d). *Technology@ Intel Magazine*, 4(10), 2007.
- [72] Tobias Holstein and Joachim Wietzke. Contradiction of separation through virtualization and inter virtual machine communication in automotive scenarios. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, page 4. ACM, 2015.
- [73] Tobias Holstein and Joachim Wietzke. Towards an Architecture for an UI-Compositor for Multi-OS Environments. In *Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings 10*, pages 138–145. Springer International Publishing, 2016.
- [74] Tobias Holstein, Markus Wallmyr, Joachim Wietzke, and Rikard Land. Current challenges in compositing heterogeneous user interfaces for automotive purposes.

- In *International Conference on Human-Computer Interaction*, pages 531–542. Springer International Publishing, 2015.
- [75] Michael Hübner and Jürgen Becker. Multiprocessor System-on-Chip - Hardware Design and Tool Integration. *Springer 2011*, 2011.
- [76] Ricky Hudi. Internationale herausforderungen in der automobilelektronik. *ATZelextronik*, 10(7):16–19, 2015.
- [77] Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, and Chul-Ryun Kim. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 257–261. IEEE, 2008.
- [78] IBM. The CoreConnect™ Bus Architecture. pages 1–8, January 1999.
- [79] Texas Instruments Incorporated. OMAP5432 EVM System Reference Guide Reference Guide. pages 1–58, May 2013.
- [80] H Inoue, A Ikeno, M Kondo, J Sakai, and M Edahiro. VIRTUS: a new processor virtualization architecture for security-oriented next-generation mobile terminals. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 484–489. IEEE, 2006.
- [81] SAE International. Sae j3061, cybersecurity guidebook for cyber-physical vehicle systems. 2016.
- [82] Asif Iqbal, Nayeema Sadeque, and Rafika Ida Mutia. An overview of microkernel, hypervisor and microvisor virtualization approaches for embedded systems. *Report, Department of Electrical and Information Technology, Lund University, Sweden*, 2110:15, 2009.
- [83] ISO. Road vehicles – functional safety – part 3. ISO 26262-3:2011, International Organization for Standardization, 2011.
- [84] ISO ISO and IEC Std. ISO 15408-1: 2009. *Information technology-Security techniques-Evaluation criteria for IT security-Part, 1*, 2009.
- [85] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.
- [86] ISO/IEC. ISO/IEC 27000. ISO/IEC, February 2016.
- [87] ITU-T. ITU-T Rec. X.1521 (04/2011) Common vulnerability scoring system, March 2012.

- [88] Yeongjin Jang, Sangho Lee, and Taesoo Kim. Breaking kernel address space layout randomization with intel tsx. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, pages 380–392, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978321. URL <http://doi.acm.org/10.1145/2976749.2978321>.
- [89] X Jin, H Chen, X Wang, Z Wang, X Wen, Y Luo, and X Li. A Simple Cache Partitioning Approach in a Virtualized Environment. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 519–524, August 2009.
- [90] Rainer Kallenbach. Trends in Automotive Electronics. In *Steinbeis Symposium, Stuttgart*, 2008.
- [91] Kernel.org. Remote processor framework, 2013. URL <https://www.kernel.org/doc/Documentation/remoteproc.txt>. [Accessed January 5st, 2018].
- [92] M Khare and A Kumar. Method and apparatus for preventing starvation in a multi-node architecture. Technical report, 2002.
- [93] Chongkyung Kil, Jinsuk Jim, C Bookholt, J Xu, and Peng Ning. Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software. *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 339–348, December 2006.
- [94] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, PACT '04*, pages 111–122, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2229-7. doi: 10.1109/PACT.2004.15. URL <http://dx.doi.org/10.1109/PACT.2004.15>.
- [95] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122. IEEE Computer Society, 2004.
- [96] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014.
- [97] DAN J KLINEDINST. Cvss and the internet of things, September 2015. URL <https://insights.sei.cmu.edu/cert/2015/09/cvss-and-the-internet-of-things.html>. [Accessed January 5st, 2018].

- [98] Andreas Knirsch, Pierre Schnarz, and Joachim Wietzke. Prioritized access arbitration to shared resources on integrated software systems in multicore environments. In *Networked Embedded Systems for Every Application (NESEA), 2012 IEEE 3rd International Conference on*, pages 1–8. IEEE, 2012.
- [99] Andreas Knirsch, Pierre Schnarz, and Joachim Wietzke. Sharb: Shared resource arbitration in partitioned multicore systems via library interposition. *International Journal of Design, Analysis and Tools for Integrated Circuits and Systems*, 4(2):18, 2013.
- [100] Andreas Knirsch, Andreas Theis, Joachim Wietzke, and Ronald Moore. Compositing User Interfaces in Partitioned In-Vehicle Infotainment. *Mensch & Computer Workshopband*, 2013.
- [101] Andreas Knirsch et al. Improved composability of software components through parallel hardware platforms for in-car multimedia systems. 2015.
- [102] P Kocher, R Lee, G McGraw, A Raghunathan, and S Ravi. Security as a new dimension in embedded system design. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 753–760. IEEE, 2004.
- [103] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *ArXiv e-prints*, January 2018.
- [104] Anthony LaMarca and Richard E Ladner. The influence of caches on the performance of sorting. *Journal of Algorithms*, 31(1):66–104, 1999.
- [105] Butler W Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.
- [106] Frédéric Leens. An introduction to i 2 c and spi protocols. *IEEE Instrumentation & Measurement Magazine*, 12(1):8–13, 2009.
- [107] Timothy E Levin, Cynthia E Irvine, Michael McEvilly, Thuy D Nguyen, et al. Separation kernel protection profile revisited: Choices and rational. 2010.
- [108] Ye Li, Richard West, and Eric Missimer. A virtualized separation kernel for mixed criticality systems. In *ACM SIGPLAN Notices*, volume 49, pages 201–212. ACM, 2014.
- [109] ARM Limited. Arm cortex-a series. Technical report, 2014.
- [110] ARM Limited. ARM System Memory Management Unit Architecture Specification SMMU architecture version 2.0. pages 1–372, July 2016.
- [111] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *ArXiv e-prints*, January 2018.

- [112] Lin Liu, Eric Yu, and John Mylopoulos. Analyzing security requirements as relationships among strategic actors. In *Submitted to the Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh, North Carolina*, 2002.
- [113] William L. Lynch, Brian K. Bray, and M. J. Flynn. The effect of page allocation on caches. In *Proceedings of the 25th Annual International Symposium on Microarchitecture, MICRO 25*, pages 222–225, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-3175-9. URL <http://dl.acm.org/citation.cfm?id=144953.145814>.
- [114] I Ripoll In-depth Marco-Gisbert, H. On the Effectiveness of Full-ASLR on 64-bit Linux. *cybersecurity.upv.es*, 2014.
- [115] Raimundas Matulevičius, Nicolas Mayer, Haralambos Mouratidis, Eric Dubois, Patrick Heymans, and Nicolas Genon. *Adapting Secure Tropos for Security Risk Management in the Early Phases of Information Systems Development*, pages 541–555. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-69534-9. doi: 10.1007/978-3-540-69534-9_40. URL http://dx.doi.org/10.1007/978-3-540-69534-9_40.
- [116] Nicolas Mayer, André Rifaut, Eric Dubois, et al. Towards a risk-based security requirements engineering framework. In *Workshop on Requirements Engineering for Software Quality. In Proc. of REFSQ*, volume 5, 2005.
- [117] Charlie McCarthy, Kevin Harnett, and Art Carter. Characterization of potential security threats in modern automobiles: A composite modeling approach. Technical report, 2014.
- [118] Rebecca T Mercuri and Peter G Neumann. Security by obscurity. *Communications of the ACM*, 46(11):160, 2003.
- [119] Roberto Mijat and Andy Nightingale. Virtualization is Coming to a Platform Near You. Technical report, ARM Limited, January 2011.
- [120] Roberto Mijat and Andy Nightingale. Virtualization is coming to a platform near you. *ARM white paper*, 20, 2011.
- [121] Charlie Miller and Chris Valasek. A survey of remote automotive attack surfaces. *Black Hat USA*, 2014.
- [122] Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle. pages 1–91, August 2015.
- [123] Milica Mitić and Mile Stojčev. An overview of on-chip buses. *Facta universitatis - series: Electronics and Energetics*, 19(3):405–428, 2006.

- [124] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: automatic detection and removal of control-flow side channel attacks. In *ICISC'05: Proceedings of the 8th international conference on Information Security and Cryptology*, pages 156–168, Berlin, Heidelberg, December 2005. Massachusetts Institute of Technology, Springer-Verlag.
- [125] Siva RK Narla. The evolution of connected vehicle technology: From smart drivers to smart cars to... self-driving cars. *Institute of Transportation Engineers. ITE Journal*, 83(7):22, 2013.
- [126] Nicolas Navet, Bertrand Delord, Markus Baumeister, et al. Virtualization in automotive embedded systems: an outlook. In *Seminar at RTS Embedded Systems*, 2010.
- [127] NIST. **National Vulnerability Database** . URL <https://nvd.nist.gov/>. [Accessed January 5st, 2018].
- [128] NSA. U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness. pages 1–182, July 2007.
- [129] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and Countermeasures - the Case of AES. *IACR Cryptology ePrint Archive*, 2005.
- [130] S Park, C T Chou, and A Kumar. Fairness mechanism for starvation prevention in directory-based cache coherence protocols. Technical report, 2012.
- [131] Anup Patel, Mai Daftedar, Mohamed Shalan, and M Watheq El-Kharashi. Embedded hypervisor xvisor: A comparative analysis. In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 682–691. IEEE, 2015.
- [132] Nick L Petroni Jr and Michael Hicks. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 103–115. ACM, 2007.
- [133] Nick Louis Petroni Jr. *Property-based integrity monitoring of operating system kernels*. ProQuest, 2008.
- [134] J. Porquet, C. Schwarz, and A. Greiner. Multi-compartment: A new architecture for secure co-hosting on soc. In *System-on-Chip, 2009. SOC 2009. International Symposium on*, pages 124–127, 2009. doi: 10.1109/SOCC.2009.5335664.
- [135] J Porquet, A Greiner, and C Schwarz. NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–4. IEEE, 2011.

- [136] Seth H Pugsley, Josef B Spjut, David W Nellans, and Rajeev Balasubramonian. SWEL: Hardware cache coherence protocols to map shared data onto shared caches. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 465–475. IEEE, 2010.
- [137] Patrick Pype, Gerardo Daalderop, Eva Schulz-Kamm, Eckhard Walters, and Maximilian von Grafenstein. Privacy and security in autonomous vehicles. In *Automated Driving*, pages 17–27. Springer, 2017.
- [138] QNX Software Systems Limited. QNC Car Platform for Infotainment, 2015. URL <http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.car.nav%2Ftopic%2Fbookset.html&cp=2>. [Accessed January 5st, 2018].
- [139] Michael Rushanan and Stephen Checkoway. Run-DMA. *9th USENIX - Workshop on offensive technologies (WOOT)*, 2015.
- [140] John Rushby. Kernels for safety. *Safe and Secure Computing Systems*, pages 210–220, 1989.
- [141] Ahmed Saeed, Ali Ahmadinia, and Mike Just. Secure on-chip communication architecture for reconfigurable multi-core systems. *Journal of Circuits, Systems and Computers*, 25(08):1650089, 2016.
- [142] F Lone Sang, Eric Lacombe, Vincent Nicomette, and Yves Deswarte. Exploiting an i/ommu vulnerability. pages 7–14, 2010.
- [143] Pierre Schnarz, Joachim Wietzke, and Ingo Stengel. Co-processor aided attack on embedded multi-os environments. In *IT Convergence and Security (ICITCS), 2013 International Conference on*, pages 1–4. IEEE, 2013.
- [144] Pierre Schnarz, Clemens Fischer, Joachim Wietzke, and Ingo Stengel. On a domain block based mechanism to mitigate DoS attacks on shared caches in asymmetric multiprocessing multi operating systems. In *ISSA*, 2014.
- [145] Pierre Schnarz, Joachim Wietzke, and Ingo Stengel. Towards attacks on restricted memory areas through co-processors in embedded multi-os environments via malicious firmware injection. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, pages 25–30. ACM, 2014.
- [146] Pierre Schnarz, Joachim Wietzke, Ingo Stengel, and Clemens Fischer. Secure separation of shared caches in amp-based mixed criticality systems. *SAIEE Africa Research Journal*, 06/2015(106):93–104, June 2015.
- [147] Pierre Schnarz, Andreas Rausch, and Joachim Wietzke. Memory-Map Shuffling: An Adaptive Security-Risk Mitigation. In *ADAPTIVE 2017*, pages 1–6, February 2017.
- [148] Bruce Schneier. Attack trees. *Dr. Dobb’s journal*, 24(12):21–29, 1999.

- [149] Markus Schumacher. LNCS 2754 - Security Engineering with Patterns. pages 1–215, December 2010.
- [150] David Seal. *ARM architecture reference manual*. Pearson Education, 2001.
- [151] Philips Semiconductors. The i2c-bus specification. *Philips Semiconductors*, 9397 (750):00954, 2000.
- [152] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, page 298, New York, New York, USA, October 2004. ACM Request Permissions.
- [153] Mary Shaw. What makes good research in software engineering? *STTT*, 2002.
- [154] Adam Shostack. *Threat modeling: Designing for security*. John Wiley and Sons, 2014.
- [155] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. Operating system concepts (9. ed.). *Wiley*, 9, 2013.
- [156] N Smith. Methods and apparatus to perform secure boot. Technical report, 2008.
- [157] Green Hills Software. Greenhills Integrity Multivisor, 2017. URL https://www.ghs.com/products/rtos/integrity_virtualization.html. [Accessed January 5st, 2018].
- [158] Dieter Spaar. Beemer, open thyself! - security vulnerabilities in bmw's connecteddrive, February 2015. URL <http://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html>. [Accessed January 5st, 2018].
- [159] ARM AMBA Specification. Multi layer AHB Specification,(rev2. 0). Technical report, 2001.
- [160] W Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall, 2010.
- [161] Ashley Stevens. Introduction to AMBA® 4 ACE™ and big.LITTLE™ Processing Technology. Technical report, July 2013.
- [162] Marius Strobl, Markus Kucera, Andrei Foeldi, Thomas Waas, Norbert Balbierer, and Carolin Hilbert. Towards automotive virtualization. In *Applied Electronics (AE), 2013 International Conference on*, pages 1–6. IEEE, 2013.

- [163] N. Suzuki, H. Kim, D. d. Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar. Coordinated bank and cache coloring for temporal protection of memory accesses. In *2013 IEEE 16th International Conference on Computational Science and Engineering*, pages 685–692, Dec 2013. doi: 10.1109/CSE.2013.106.
- [164] David Tam, Reza Azimi, Livio Soares, and Michael Stumm. Managing shared l2 caches on multicore systems in software. In *In Proc. of the Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA)*, 2007.
- [165] B. Tan, M. Biglari-Abhari, and Z. Salcic. A system-level security approach for heterogeneous mpsocs. In *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 74–81, Oct 2016. doi: 10.1109/DASIP.2016.7853800.
- [166] Benjamin Tan, Morteza Biglari-Abhari, and Zoran Salcic. Towards decentralized system-level security for mpsoc-based embedded applications. *Journal of Systems Architecture*, 80(Supplement C):41 – 55, 2017. ISSN 1383-7621. doi: <https://doi.org/10.1016/j.sysarc.2017.09.001>. URL <http://www.sciencedirect.com/science/article/pii/S1383762117300322>.
- [167] Andrew S Tanenbaum and Albert S Woodhull. *Operating systems: design and implementation*, 2014.
- [168] PaX Team. Pax address space layout randomization (aslr), 2003. URL <http://pax.grsecurity.net/docs/pax.txt>. [Accessed January 5st, 2018].
- [169] Texas Instruments. OMAP5432 Multimedia Device - Silicon Revision 2.0 Evaluation Module. 2012.
- [170] Texas Instruments Incorporated. OMAP5432 Processor-based EVM. URL http://www.ti.com/diagrams/med_omap5432-evm_omap5432_evm_2.jpg. [Accessed January 5st, 2018].
- [171] Texas Instruments Incorporated. Software Development Kits (Evaluation-SW) for OMAP5432 processor-based EVM, 2013. URL <http://www.ti.com/tool/omap5432-evm-eval-sw>. [Accessed January 5st, 2018].
- [172] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation. <http://www.commoncriteriaportal.org/>, September 2006. URL <http://www.commoncriteriaportal.org/>.
- [173] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation, Evaluation methodology. <http://www.commoncriteriaportal.org/>, 2017. URL <http://www.commoncriteriaportal.org/>.

- [174] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components. <http://www.commoncriteriaportal.org/>, 2017. URL <http://www.commoncriteriaportal.org/>.
- [175] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components. <http://www.commoncriteriaportal.org/>, 2017. URL <http://www.commoncriteriaportal.org/>.
- [176] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model. <http://www.commoncriteriaportal.org/>, 2017. URL <http://www.commoncriteriaportal.org/>.
- [177] TI, Android Open Source Project. OMAP5 development platform - git repository, 2013. URL <http://git.omapzoom.org/?p=device/ti/omap5sevm>. [Accessed January 5st, 2018].
- [178] He Yeping Tian Shuo and Ding Baozeng. LNCS 7232 - Prevent Kernel Return-Oriented Programming Attacks Using Hardware Virtualization. pages 1–12, March 2012.
- [179] Lionel Torres, Pascal Benoit, Gilles Sassatelli, Michel Robert, Fabien Clermidy, and Diego Puschini. An Introduction to Multi-Core System on Chip – Trends and Challenges. In *Multiprocessor System-on-Chip*, pages 1–21. Springer New York, New York, NY, 2011.
- [180] G M Uchenick and W M Vanfleet. Multiple independent levels of safety and security: high assurance architecture for MSLS/MLS. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 610–614. IEEE, 2005.
- [181] Prathap Kumar Valsan, Heechul Yun, and Farzad Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, pages 1–12. IEEE, 2016.
- [182] W Mark Vanfleet, R William Beckwith, Ben Calloni, Jahn A Luke, Carol Taylor, and Gordon Uchenick. MILS:Architecture for High-Assurance Embedded Computing. pages 1–5, July 2005.
- [183] Prashant Varanasi and Gernot Heiser. Hardware-supported virtualization on arm. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, pages 11:1–11:5, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1179-3. doi: 10.1145/2103799.2103813. URL <http://doi.acm.org/10.1145/2103799.2103813>.

- [184] John Viega, Tadayoshi Kohno, and Bruce Potter. Trust (and mistrust) in secure applications. *Commun. ACM*, 44(2):31–36, February 2001. ISSN 0001-0782. doi: 10.1145/359205.359223. URL <http://doi.acm.org/10.1145/359205.359223>.
- [185] Wind River Systems, Inc. WindRiver Virtualization, 2017. URL <https://www.windriver.com/products/operating-systems/virtualization/>. [Accessed January 5st, 2018].
- [186] W Wolf, A A Jerraya, and G Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.
- [187] William Wulf, Ellis Cohen, William Corwin, Anita Jones, Roy Levin, Charles Pierson, and Fred Pollack. Hydra: The kernel of a multiprocessor operating system. *Communications of the ACM*, 17(6):337–345, 1974.
- [188] Ben-Ami Yassour, Muli Ben-Yehuda, and Orit Wasserman. On the DMA mapping problem in direct device assignment. *the 3rd Annual Haifa Experimental Systems Conference*, page 18, May 2010.
- [189] Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. COLORIS - a dynamic cache partitioning system using page coloring. *PACT*, 2014.
- [190] R K Yin. *Case Study Research: Design and Methods*. SAGE Publications, 2013.
- [191] Xiao Zhang, Sandhya Dwarkadas, and Kai Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 89–102, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-482-9. doi: 10.1145/1519065.1519076. URL <http://doi.acm.org/10.1145/1519065.1519076>.



A

A.1 Schematics



Fig. A.1 TI OMAP5 IP-cores and Communication Architecture [170].

A.2 Features

The Cortex-A15 subsystem [169, p. 292f]

- ARM Cortex-A15 MPCore
 - Two central processing units (CPUs)
 - ARM Version 7 ISA: Standard ARM instruction set plus Thumb®-2, Jazelle® RCT Java™ accelerator, hardware virtualization support, and large physical address extensions (LPAE)
 - Neon™ SIMD coprocessor and VFPv4 per CPU
 - Interrupt controller with up to 160 interrupt requests
 - One general-purpose timer and one watchdog timer per CPU
 - Debug and trace features

Cache Subsystem [169, p. 1058]

- 32-KiB L1 instruction (L1I) and 32-KiB L1 data (L1D) cache
 - 64-byte line size
 - 2-way set associative
- Shared 2-MiB level 2 (L2) cache
 - unified (instructions and data)
 - cache organized as 16 ways of 2048 sets of 64-byte

Memory management unit (MMU) [169, p. 1059]

- Two-level translation lookaside buffer (TLB) organization
- Second level is a unified, 4-way associative, 512-entry main TLB
- First level is an 32-entry, fully associative micro-TLB implemented for each of instruction fetch, load, and store.
- Supports hardware TLB table-walk for backward-compatible and new 64-bit entry page table
- New page table format can produce 40-bit physical addresses

- Two-stage translation where first stage is HLOS-controlled and the second level may be controlled by a hypervisor. Second stage always uses the new page table format

