

Adaptive Isolation for Predictability and Security

Edited by

Tulika Mitra¹, Jürgen Teich², and Lothar Thiele³

1 National University of Singapore, SG, tulika@comp.nus.edu.sg

2 Friedrich-Alexander-Universität Erlangen-Nürnberg, DE,
teich@informatik.uni-erlangen.de

3 ETH Zürich, CH, thiele@ethz.ch

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 16441 “Adaptive Isolation for Predictability and Security”. Semiconductor technology is at the verge of integrating hundreds of processor cores on a single device. Indeed, affordable multi-processor system-on-a-chip (MPSoC) technology is becoming available. It is already heavily used for acceleration of applications from domains of graphics, gaming (e.g., GPUs) and high performance computing (e.g., Xeon Phi). The potential of MPSoCs is yet to explode for novel application areas of embedded and cyber-physical systems such as the domains of automotive (e.g., driver assistance systems), industrial automation and avionics where non-functional aspects of program execution must be enforceable. Instead of best-effort and average performance, these real-time applications demand timing predictability and/or security levels specifiable on a per-application basis. Therefore the cross-cutting topics of the seminar were methods for temporal and spatial isolation. These methods were discussed for their capabilities to enforce the above non-functional properties without sacrificing any efficiency or resource utilization. To be able to provide isolation instantaneously, e.g., even for just segments of a program under execution, adaptivity is essential at all hardware- and software layers. Support for adaptivity was the second focal aspect of the seminar. Here, virtualization and new adaptive resource reservation protocols were discussed and analyzed for their capabilities to provide application/job-wise predictable program execution qualities on demand at some costs and overheads. If the overhead can be kept low, there is a chance that adaptive isolation, the title of the seminar, may enable the adoption of MPSoC technology for many new application areas of embedded systems.

Seminar October 30–4, 2016 – <http://www.dagstuhl.de/16441>

1998 ACM Subject Classification C.3 Special-Purpose and Application-Based Systems: Real-time and embedded systems

Keywords and phrases Adaptive isolation, Embedded systems, Real-Time systems, Predictability, Security, MPSoC, Parallel computing, Programming models, Timing analysis, Virtualization

Digital Object Identifier 10.4230/DagRep.6.10.120



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Adaptive Isolation for Predictability and Security, *Dagstuhl Reports*, Vol. 6, Issue 10, pp. 120–153

Editors: Tulika Mitra, Jürgen Teich, and Lothar Thiele



DAGSTUHL
REPORTS

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Tulika Mitra

Jürgen Teich

Lothar Thiele

License © Creative Commons BY 3.0 Unported license
© Tulika Mitra, Jürgen Teich, and Lothar Thiele

Semiconductor industry has shifted from processor clock speed optimization (having reached its physical limits) to parallel and heterogeneous many-core architectures. Indeed, the continuous technological scaling enables today the integration of hundred and more cores and, thus, enormous parallel processing capabilities. Whereas higher (average) performance has been and still is the major driver for any MPSoC platform design, there is a huge hesitation and fear to install such platforms in embedded systems that require predictable (boundable) guarantees of non-functional properties of execution rather than average properties for a mix of applications. Moreover, it may be observed that in an embedded system, each application running on a platform typically a) requires different qualities to be satisfied. For example, one application might demand for authentication, thus requiring the guarantee of unmodified data and program but have no requirements on speed of execution. Another application might rather require the execution to meet a set of real-time properties such as a deadline or a target data rate. To give an example, consider a driver assistance video processing application in a car that must detect obstacles in front of the car fast enough so to activate the brake system in a timely manner. It must therefore be possible to enforce a set of non-functional qualities of execution on a multi-core platform on a per-application/job basis. b) The above requirements on execution qualities may even change over time or during the program execution of a single application or being dependent on user or environmental settings. For example, one user might not care about sending or distributing personal information over the communication interfaces of a mobile phone whereas another one cares a lot, even in the presence of side channels.

Unfortunately, the way MPSoCs are built and programmed today, the embedded system engineers often experience even worse execution qualities than in the single core case, the reason being the sharing of resources such as cores, buses and/or memory in an unpredictable way. Another obstacle for a successful deployment of multi-core technology in embedded systems is the rather unmanageable complexity. This holds particularly true for the analysis complexity of a system for predictable execution qualities at either compile-time or run-time or using hybrid analysis techniques. The complexity is caused here by an abundant number of resources on the MPSoC and the increasing possibilities of interference created by their concurrent execution and multiple layers of software controlling program executions on a platform. Such layers are often designed for contradictory goals. For example, the power management firmware of an MPSoC may be designed to reduce the energy/power consumption or avoid temperature hot spots. The OS scheduler, on the other hand, may be designed to maximize the average CPU utilization for average performance. Providing tight bounds on execution qualities of individual applications sharing an execution platform is therefore not possible on many MPSoC platforms available today.

One remedy out of this dilemma that has been proposed a long time before the introduction of any MPSoC technology is isolation. With isolation, a set of techniques is subsumed to separate the execution of multiple programs either spatially (by allocating disjoint resources) or temporally (by separating the time intervals shared resources are used). Additionally, in order to provide isolation on demand, there is the need for adaptivity in all hardware as

well as software layers from application program to executing hardware platform. Indeed, adaptivity is considered a key topic in order to reduce or bound execution quality variations actively on a system and in an on-demand manner for the reason to neither overly restrict nor to underutilize available resources.

Adaptive Isolation, the topic of the proposed Dagstuhl seminar, may be seen as a novel and important research topic for providing predictability of not only timing but also security and may be even other properties of execution on a multi-core platform on a per application/job basis while easing and trading off compile-time and run-time complexity.

First, a common understanding of which techniques may be used for isolation including hardware units design, resource reservation protocols, virtualization techniques, and including novel hybrid and dynamic resource assignment techniques were discussed. Second, a very interdisciplinary team of experts including processor designers, OS and compiler specialists, as well as experts for predictability and security analysis were brought together for evaluating these opportunities and presenting novel solutions. The competencies, experiences, and existing solutions of the multiple communities stimulated discussions and co-operations that hopefully will manifest in innovative research directions for enabling predictability on demand on standard embedded MPSoCs.

2 Table of Contents

Executive Summary	
<i>Tulika Mitra, Jürgen Teich, and Lothar Thiele</i>	121
Major Topics Discussed	125
Adaptive Isolation for Timing Predictability	125
Isolation and Adaptivity for Security	125
Cross-Cutting Concerns	126
Summary of the Presentations	126
Predictability	128
Security	130
Cross-cutting Concerns for Adaptive Isolation	130
Abstract of Talks	131
Network-on-Chip-Assisted Adaptive Partitioning and Isolation Technology for “Dynamic” Homogeneous Manycores <i>Davide Bertozzi and Balboni Marco</i>	131
Use only when you need – Providing adaptive temporal isolation in Cyber-Physical Systems <i>Samarjit Chakraborty</i>	132
Achieving Timing Predictability by Combining Models <i>Heiko Falk and Arno Luppold</i>	133
Soteria: Offline Software Protection within Low-cost Embedded Devices <i>Johannes Götzfried</i>	134
Challenges of Temporal Isolation <i>Gernot Heiser</i>	134
Predictability in Multicore Systems Using Self-Suspending Tasks <i>Jian-Jia Chen</i>	135
Software Development for Isolation <i>Tulika Mitra</i>	135
Time-Based Intrusion Detection in Cyber-Physical Systems <i>Frank Mueller</i>	136
Adaptive Pipeline for Security in Real Time Systems <i>Sri Parameswaran</i>	136
Timing Predictability and How to Achieve It <i>Jan Reineke</i>	137
Connecting the dots – Towards the total automation of embedded systems design (in Java world) <i>Zoran Salcic</i>	138
Isolation for Security <i>Patrick Schaumont</i>	138

T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems <i>Martin Schoeberl</i>	139
Adaptive Memory Protection for Many-Core Systems <i>Wolfgang Schröder-Preikschat</i>	140
Security Issues on the Boundary of Ideal and Real Worlds <i>Takeshi Sugawara</i>	141
An Introduction to the Seminar <i>Jürgen Teich</i>	141
Isolation, resource efficiency and covert channels <i>Lothar Thiele</i>	142
Determinate and Timing-Predictable Concurrency in Reactive Systems – The Synchronous Approach and the SCCharts Language <i>Reinhard von Hanxleden</i>	142
Hybrid Application Mapping for Dynamic Isolation in Invasive Computing <i>Stefan Wildermann</i>	143
Timing Verification – Flogging a Dead Horse? <i>Reinhard Wilhelm</i>	143
Working groups	144
Runtime Monitoring <i>Felix Freiling</i>	144
Future of Timing Verification <i>Samarjit Chakraborty</i>	147
Attack Models <i>Albert Cohen and Karine Heydemann</i>	148
Synergy between Predictability and Security <i>Frank Mueller</i>	149
Models of Computation and Programming Languages <i>Reinhard von Hanxleden</i>	150
Panel Discussion	152
Acknowledgements	152
Participants	153

3 Major Topics Discussed

In the following, major topics and questions that were raised and discussed during the seminar, are summarized.

3.1 Adaptive Isolation for Timing Predictability

- New ways to establish isolation by means of hardware and software: Which of the approaches and known concepts for isolation can be used in adaptive scenarios, which rather not?
- Analysis: Statistical vs. hard guarantees? What are limitations of either approach? Can these techniques be reasonably generalized to other architectural elements besides caches?
- Hybrid Analysis and Resource Management: Novel techniques for Mixed Static/Dynamic Resource Assignment and Analysis. Which improvements (e.g., reduced search space vs. less pessimistic bounds) may these techniques deliver and for which set of applications (e.g., periodic streaming, DSP, aperiodic real-time control, mixed critical applications) may these be applied? How may the search space for design decisions regarding resource assignment and scheduling be reduced to a minimum through a characterization of static vs. run-time?
- Online isolation through reconfiguration (e.g., dynamic TDMA adaptation, switching protocols, dynamic schedule adaptation).
- Adaptive hardware architectures (e.g., processor buses with switchable protocols: static priority vs. TDMA depending on workload mix at run-time).
- Utilization and timing analysis for unknown execution time and workload scenarios.
- Cost/Benefit Analysis of adaptive isolation techniques: How much more expensive are adaptive techniques in relation to conventional techniques (Hardware/Software Overheads, adaptation time (e.g., switching times, optimization times, times until stabilization, utilization gains, etc.).
- How can we bound the interference between tasks due to heat transfer?

3.2 Isolation and Adaptivity for Security

- Definition of security in an adaptive MPSoC context. How do security issues change by introducing adaptivity? What is the attackers' model?
- Security bottlenecks of current MPSoC systems with respect to hardware architecture and the possibilities to isolate applications.
- Security requires a root of trust. Security also makes use of isolation. We should reason about secure hand-over in the context of adaptivity. When software modules move from one hardware unit to another one, how are the root of trust and isolation transferred?
- With respect to which properties may security be defined? For example, basic isolation might be defined as a guarantee that no other application may read or write the data of another. For example, a designer or user of an app might require that the data entered or processed to be confidential or request a guarantee that it is unaltered.
- Which techniques must be available at the hardware and software side to enforce certain levels of security on a per-application basis on an MPSoC platform and what is the expected overhead of such techniques?
- May different levels of per-application/job security also be established adaptively?
- Hardware architecture designs for adaptive security.

- Do there exist other levels of security? For example, side channel attacks? Which isolation techniques may be employed on an MPSoC to restrict, prevent, or minimize the chances of attacks, e.g., in terms of resource isolation through resource allocation techniques, encryption on demand on a Network-on-Chip, etc.?
- Is heat transfer a side-channel information leakage source? Can it be a threat to privacy and security? How can we quantify the corresponding effects and what are reasonable countermeasures?

3.3 Cross-Cutting Concerns

From a resource management's point of view, modern embedded system applications come with significant challenges: Highly dynamic usage scenarios as already observable in today's "smart devices" result in a varying number of applications, each with different characteristics, but running concurrently at different points in time on a platform. Obviously, full isolation, avoiding any resource sharing (e.g., by partitioning) is generally too costly or inefficient (utilization). No isolation, on the other hand, will not allow for timing and security properties to hold. From the programmer's point of view, strong isolation and efficient sharing are desired, but they represent two opposing goals.

Traditional techniques to provide a binding or pinning of applications to processors are either applied at design time and result in a static system design. Such a static design may, on the one hand, be too optimistic by assuming that all assigned resources are always available or it may require for over-allocation of cores to compensate for worst-case scenarios.

In this area, cross-cutting techniques such as partitioning, gang scheduling, dynamic resource allocation, virtualization, e.g., real-time scheduling in hypervisors, are opportunities that were discussed for their capability for providing some degree of isolation and capabilities of providing quality on demand per application/job.

Finally, the interaction between security and timing predictability were explored. A malware can compromise a real-time system by making an application miss its deadline and the system should ensure that deadline overruns in the presence of malware be predicted early and remedial actions taken. On the other hand, as the bounds on execution times of an application are known in real-time systems, an execution time outside the bound indicates the possibility of unauthorized code execution and provides an additional mechanism for malware detection. The scheduling and resource allocation should also take into account the trade-off between the timing overheads of security protection mechanism (e.g., encryption cost) leading to increased execution time (and hence difficulty in schedulability) vis-à-vis the need for security isolation.

4 Summary of the Presentations

The presentations in this seminar included state-of-the-art adaptive isolation techniques for both security and predictability. Five breakout sessions covered discussions on the current status, future challenges and research opportunities. A very interdisciplinary team of experts including processor designers, OS and compiler specialists, as well as experts on predictability and security evaluated these opportunities and presented novel solutions. This subsection presents an overview of the topics covered by individual speakers in the seminar. Please refer to the included abstracts to learn more about the individual presentations.

The seminar opened with an introduction by organizer Jürgen Teich (Friedrich-Alexander-Universität Erlangen-Nürnberg). He explained the motivation behind the seminar in the

context of emerging many-core architectures. These architectures can potentially be deployed in embedded systems with strict timing and security guarantee requirements. He presented different definitions of timing predictability and sources of unpredictability such as resource sharing, multi-threading, and power management. He briefly talked about adaptive isolation techniques, such as resource reservation and virtualization, developed in the context of the Invasive Computing (InvasIC) project¹ – a DFG-funded Transregional Collaborative Research Center investigating a novel paradigm for the design and resource-aware programming of future parallel computing systems. He emphasized the similarities between the adaptive isolation techniques for security and timing predictability – a key theme of the seminar.

The introduction was followed by two keynote talks: one on predictability and one on security. The predictability keynote was delivered by Jan Reineke (Universität des Saarlandes). He explained two sources of variation in execution time for software timing analysis: the program input and the micro-architectural state. He raised the key concern that interference due to resource sharing (for L2 cache and memory controller for example) can lead to significant slowdown on multi-core platform compared to single-threaded execution. He stressed the importance of deriving accurate timing models given the lack of information available regarding timing in the underlying architecture. He defined predictability and analyzability as two important but different properties essential towards accurate software timing analysis. Both predictability and analyzability can be enhanced by eliminating stateful micro-architectural components by stateless ones (e.g., replacing caches with scratchpad memory), eliminating interference in shared resources through isolation, and choosing “forgetful” micro-architectural components (e.g., Pseudo LRU replacement policy in place of LRU). In addition, analyzability can be improved if the underlying platform exhibits freedom from timing anomalies (local worst case does not lead to global worst case in systems with timing anomaly) and offers timing compositionality. He presented strictly in-order pipeline processors as an example of such an ideal platform; but the performance impact of such an architecture and its commercial viability remain unknown.

The keynote talk on security was delivered by Patrick Schaumont (Virginia Polytechnic Institute). He motivated the need for secure isolation by introducing a contemporary trusted medical application where privacy/security mechanisms need to be enforced in an end-to-end fashion from tiny micro-controllers (for sensing) to more powerful multi-cores (for gateway device) and finally to servers with complex processors, large memory, and huge storage (for data analytics). He emphasized the key concerns in such platforms, namely, security, safety, and privacy, that demand isolated storage, communication, and execution. The two building blocks of secure computing are the trust boundary and the attacker models that breach the trust boundaries. Isolation is one (but not the only) way to achieve trust by providing confidentiality guarantees in a secure implementation. However, it is important to remember that complete isolation is not a feasible alternative and isolation for security almost always incurs overhead either in terms of area or performance just like predictability. He then presented two examples of isolation for security: SANCUS for lightweight isolation in micro-controllers and SGX for server class isolation. In closing, Patrick mentioned few open challenges such as quantifying security and its resource overhead through well-defined metrics and classifying properties of secure computing in general and secure computer architectures in particular.

¹ <http://www.invasic.de>

4.1 Predictability

The topics covered under adaptive isolation for predictability centered around the future of predictability, design of predictable architectures, providing isolation in general-purpose multi-/many-core architecture, and predictability in reactive systems.

4.1.1 Future of predictability

The talks on timing predictability presented two contrasting views. Reinhard Wilhelm (Universität des Saarlandes) concurred with Jan Reineke’s viewpoint in the keynote that predictability and analyzability are becoming increasingly challenging and even impossible with continuous advances in commercial micro-architectures that harm rather than aid predictability. Architectural complexity leads to analysis complexity. The recipe for success in timing analysis has been abstraction and decomposition. Unfortunately, contemporary processors – even processors supposed to be designed for real-time systems (such as ARM Cortex R5F) – include features (e.g., random replacement caches) that make abstraction and decomposition infeasible. Alternatives to static timing analysis, such as measurement-based methods, do not offer soundness and at the same time suffer from lack of accurate timing models just like static analysis.

In contrast to these views that embedded systems require complete timing predictability, Samarjit Chakraborty (TU München) claimed that in certain applications, such as control systems, it is possible to live with less than total timing predictability. As most controllers exhibit certain degree of “robustness”, the behavior of the controller will not be impacted if some deadlines are missed. Thus, timing analysis, instead of focusing on deadline constraints, should focus on higher-level (control theoretic) goals that better characterize system performance requirements. Achieving this, however, requires quantifying the robustness of the controller to identify the deadlines that are crucial to be satisfied and the ones that can be ignored without any major impact on controller outcome.

4.1.2 Predictable Architecture and Optimizations

Reinhard Wilhelm presented a constructive approach called PROMPT architecture that provides timing isolation for each task when executing multiple tasks on a multi-core architecture. The generic PROMPT architecture is instantiated for each application so as to minimize interferences among the tasks of the application as much as possible. The idea of time predictable architecture was also revisited by Martin Schoeberl (Technical University of Denmark) who presented T-CREST, a time-predictable multi-core architecture for embedded systems. The vision behind T-CREST is to make the worst-case fast and the whole system analyzable rather than make the common case fast as is the conventional wisdom in general-purpose architecture community. The architecture provides constant-time execution of instructions, time-division-multiplexing in the Network-on-Chip (NoC), and software-controlled scratchpad memory for predictability. More importantly, T-CREST provides a complete platform implemented in FPGAs as well as simulator supporting both compiler and analysis tools released under open source BSD license.

Zoran Salcic (University of Auckland) presented an orthogonal solution for timing predictability starting from formal specification of the system in SystemJ, which is based on a formal model of computation. The key feature of SystemJ is Globally Asynchronous Locally Synchronous (GALS) model while incorporating Java for objects and computations allowing SystemJ specification to be executable on any Java processor. He presented an automated design flow that can translate the formal specification to custom NoC-based heterogeneous

multiprocessor platform through design space exploration, optimizations, and scheduling. This is similar in vein to the PROMPT approach, except that software code, schedule and platform instance are all generated automatically in this approach. He concluded his talk by demonstrating an automated bottling machine designed with this model-driven approach.

Heiko Falk (TU Hamburg-Harburg) presents his vision to achieve predictability by combining models during compilation. In current software design practice for real-time systems, the software is designed and optimized for the average-case behavior followed by software timing analysis to ensure that the execution meets deadline constraints. He proposed design of WCET-aware compiler that optimizes software for the worst-case execution time rather than average case. This is achieved by integrating the timing models initially designed for analysis in the compiler itself. Combined with more predictable architectural components, such as scratchpad memory instead of cache, the WCC compiler can provide resource isolation and enables schedulability in some systems that could not meet deadlines under existing software design process.

4.1.3 Isolation for predictability in multi-core

Stefan Wildermann (Universität Erlangen-Nürnberg) followed up from the introduction by Jürgen Teich on achieving adaptive isolation in the context of Invasive Computing. He described a hybrid mapping approach where the solution for each individual task is obtained statically but these individual solutions are put together at runtime through a compositional timing analysis that relies on a composable NoC. The main idea is to carry out performance analysis and design space exploration for individual tasks at design time and identify a set of Pareto-optimal points. At runtime, depending on the scenario, a set of design points (one per task) are identified that satisfy the constraints for all the tasks. The downside of this approach is the huge runtime to choose these design points and may outweigh the benefits of isolation.

Jian-Jia Chen (TU Dortmund) focused on system-level timing analysis in multi-core systems with multiple tasks. Current two-phase analysis approaches find the WCET of each task individually and then compute the worst-case response time (WCRT) of a set of tasks by considering interference from other tasks for shared resources. However, in the presence of shared resources, a task might be suspended from execution when it cannot get immediate access to the resource. This self-suspension of tasks needs to be accounted for in WCRT analysis. But many existing works fail to handle the impact of self-suspension correctly leading to overly optimistic execution time. His talk pointed out the challenges in providing predictability on multi-cores: isolation through time-division-multiplexing introduces unnecessary pessimism and cannot work if the tasks need to share information. On the other hand, with sharing, WCRT analysis and schedulability tests are not well-equipped to handle the interference that need synergy between scheduler design, program analysis, and models of computation.

4.1.4 Reactive Systems

Reinhard von Hanxleden (Universität Kiel) and Albert Cohen (ENS-Paris) discussed predictability in reactive systems. Reinhard von Hanxleden talked about the power of synchronous programming languages such as SCADE and SCCharts. He showed the extensions to these languages that allow deterministic implementation in hardware/software directly from the model. He emphasized the importance of compilation approach on timing predictability, specially in model-to-model mappings. Albert Cohen presented control systems with significant

computations and how to reconcile the computation with the control. A synchronous language like Lustre is extended with isolation control features and ability to safely accommodate some delay in the computation. Similar to Reinhard von Hanxleden’s approach, the compiler plays crucial role in mapping the abstract model and real-time scheduling onto multi-core system with simple runtime support for adaptive isolation.

4.2 Security

In his keynote, Patrick Schaumont talked about hardware architectures needed for secure isolation. Johannes Götzfried (Universität Erlangen-Nürnberg) presented Soteria – a lightweight solution for secure isolation in low-cost embedded devices. Soteria can effectively protect the confidentiality and integrity of an application against all kinds of software attacks including attacks from the system level. Soteria achieves this through a simple program-counter based memory access control extension for the TI MSP430 microprocessor with minimal overhead.

Tulika Mitra (National University of Singapore) mentioned the challenges associated with the adoption of secure isolation mechanisms by software developers. She presented an automated approach that, given an Android application, can identify the sensitive code fragments, move them to the secure world (ARM TrustZone), and finally re-factor the original application to establish communication between the normal code fragments and the secure code fragments. This automation takes away the burden of utilizing secure isolation mechanisms by software developers.

Lothar Thiele (ETH Zürich) introduced the possibility of thermal covert channels in multi-core systems. He demonstrated that the on-chip temperature sensors can represent a security breach by allowing otherwise isolated applications running on different cores to communicate and possibility leak sensitive data. A quantification of the covert channel capacity leveraging both theoretical results from information theory and experimental data from modern platforms (such as Android phone) showed sufficient bandwidth for the channel to be useful for information leakage.

Sri Parameswaran (UNSW Sydney) presented an online monitoring technique to detect and recover from hardware Trojans in pipelined multiprocessor system-on-chip devices. The system adapts and randomizes to provide security. Takeshi Sugawara (Mitsubishi, Kanagawa) stressed the importance of assumptions (model abstractions) in security. In the context of side-channel attacks and LSI reverse engineering, he showed how the countermeasures are constructed and how their assumptions are falsified. He also presented static isolation based on domain-specific coprocessors.

4.3 Cross-cutting Concerns for Adaptive Isolation

Adaptive isolation mechanisms that can be employed for both security and predictability, as well as the synergy and conflict between security and predictability featured prominently and repeatedly in the seminar.

Gernot Heiser (UNSW Sydney) pointed out the challenges towards isolation from both security and predictability perspective. He opined that spatial isolation is relatively easy to achieve, both in single-core and multi-core settings, given the support from both hardware and software. He cited seL4 micro-kernel as an example to illustrate his point. He, however, re-iterated (just like Reinhard Wilhelm and Jan Reineke) that temporal isolation is much harder especially in the presence of complex, unpredictable hardware. The instruction-set

architecture (ISA) no longer provides a guaranteed contract between hardware and software for either timeliness or security (for example, hidden states and timing channels). He called on the architects to extend the ISA so that timing effects become visible and hardware provides mechanisms to partition or flush shared states with bounded latency so as to provide isolation.

Davide Bertozzi (Università di Ferrara) focused on NoC to provide adaptive isolation in many-core architectures. He had a different opinion from Gernot Heiser regarding spatial isolation and showed that current many-core accelerator architectures are at odds with spatial-division multiplexing. For example, the traffic generated by different applications may collide in the NoC when NoC paths are shared between nodes assigned to different applications even if each core is allocated to only a single application exclusively. He presented routing restrictions as an approach towards partitioning the resources among different applications; but this leads to additional challenges in mapping as well as reconfigurability and adaptivity of the partitions.

Wolfgang Schröder-Preikschat (Universität Erlangen-Nürnberg) presented isolation in memory to protect against unintentional programming errors as well as attacks from malicious programs/processes. While existing memory-management units provide protection, they harm time predictability. There are scenarios where ubiquitous memory protection is unnecessary and increases uncertainty for some time, but is required at other points during the run time of a system. He proposed adaptive memory protection as a solution, where the protection state of applications can change over time. It allows the combination of benefits of both worlds: security when memory protection is needed and increased performance and predictability once security is superfluous.

Sibin Mohan (University of Illinois at Urbana-Champaign) expounded on the interaction between predictability and security. He alerted the audience to the challenges of real-time systems running in insecure world. Real-time systems demand predictability; but predictability actually enables the attackers to precisely reconstruct the execution behavior of the system. In particular, he showed how information about the behavior of real-time systems (e.g., schedule) can be leaked by adversaries and presented techniques to deter such attacks. On the other hand, sometimes it is possible to use the predictable behavioral properties of real-time systems to actually detect intrusion almost as soon as they occur. Frank Mueller (North Carolina State University) also exploited the synergy between predictability and security. His approach utilizes the timing bounds obtained for different code fragments of a program during static timing analysis. At runtime, if the execution time of a code fragment falls outside its pre-determined bounds, the system flags an intrusion

5 Abstract of Talks

5.1 Network-on-Chip-Assisted Adaptive Partitioning and Isolation Technology for “Dynamic” Homogeneous Manycores

Davide Bertozzi (Università di Ferrara, IT) and Balboni Marco

License © Creative Commons BY 3.0 Unported license
© Davide Bertozzi and Balboni Marco

Joint work of Davide Bertozzi, Marco Balboni, Giorgos Dimitrakopoulos, José Flich

The software parallelism is not keeping up with hardware parallelism, therefore the problem of efficiently exploiting large array fabrics of homogeneous processing cores will soon come

to the forefront. Multi-programmed mixed-criticality workloads are the straightforward solution to this problem, although they raise a number of practical issues ranging from system composability techniques for easier verification, to performance predictability and/or security. This talk presents a systematic approach to these issues through an adaptive partitioning and isolation technology for manycore computing fabrics having its key enabler in the reconfigurable features of the on-chip interconnection network. The technology relies on two main pillars. First, a hierarchy of partition types enables to properly sandbox applications with controlled degrees of interactions and/or dependencies (if at all allowed). Second, fine-grained adaptivity of the system configuration to the workload is implemented with NoC assistance for the sake of power-efficient resource management at any given point in time. It follows from this a “design-for-partitioning” philosophy that is at the core of future dynamic hardware platforms, and that will shape their architectures from the ground up.

5.2 Use only when you need – Providing adaptive temporal isolation in Cyber-Physical Systems

Samarjit Chakraborty (TU München, DE)

License © Creative Commons BY 3.0 Unported license
© Samarjit Chakraborty

Joint work of Samarjit Chakraborty, Alejandro Masrur, Ansuman Banerjee, Anuradha M. Annaswamy, Jian-Jia Chen, Dip Goswami, Harald Voit, Reinhard Schneider

Main reference A. Masrur, D. Goswami, S. Chakraborty, J.-J. Chen, A. Annaswamy, A. Banerjee, “Timing analysis of cyber-physical applications for hybrid communication protocols”, in Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2012), pp. 1233–1238, IEEE, 2012.

URL <http://dx.doi.org/10.1109/DATE.2012.6176681>

Many embedded control systems have distributed implementations, in which sensor values and control signals have to be communicated over shared communication buses. The participants sharing the bus along with the bus protocol being used determine the delay suffered by the control signals, which in turn affect stability and control performance. Two broad classes of communication protocols exist, which are based on either the time-triggered or the event-triggered paradigms. The former ensures strict temporal isolation between messages and results in more deterministic communication. Hence, it is easier to use when guarantees on stability and control performance are required. The latter does not provide temporal isolation between messages, but has several advantages like better bus utilization and easier extensibility. This has also resulted in hybrid protocols that combine the event- and time-triggered paradigms. However, there has been little work on how to exploit such hybrid protocols when designing control algorithms, in order to utilize the benefits of both the communication paradigms. In this talk we will discuss this problem and propose some promising research directions that involve adaptively providing temporal isolation on a when-needed basis. This brings up a number of challenges both in the areas of control theory, and also in timing analysis.

References

- 1 Harald Voit, Anuradha M. Annaswamy, Reinhard Schneider, Dip Goswami, Samarjit Chakraborty. *Adaptive switching controllers for systems with hybrid communication protocols*. American Control Conference (ACC) 2012
- 2 Harald Voit, Anuradha Annaswamy, Reinhard Schneider, Dip Goswami, Samarjit Chakraborty. *Adaptive switching controllers for tracking with hybrid communication protocols*. 51th IEEE Conference on Decision and Control (CDC) 2012

- 3 Alejandro Masrur, Dip Goswami, Samarjit Chakraborty, Jian-Jia Chen, Anuradha Anaswamy, Ansuman Banerjee. *Timing analysis of cyber-physical applications for hybrid communication protocols*. Design, Automation & Test in Europe Conference (DATE) 2012
- 4 Dip Goswami, Reinhard Schneider, Samarjit Chakraborty. *Re-engineering cyber-physical control applications for hybrid communication protocols*. Design, Automation & Test in Europe Conference (DATE) 2011

5.3 Achieving Timing Predictability by Combining Models

Heiko Falk (TU Hamburg-Harburg, DE) and Arno Luppold

License © Creative Commons BY 3.0 Unported license
© Heiko Falk and Arno Luppold

Main reference A. Luppold, H. Falk, “Code Optimization of Periodic Preemptive Hard Real-Time Multitasking Systems”, in Proc. of the 18th Int’l Symposium on Real-Time Distributed Computing (ISORC 2015), pp. 35–42, IEEE, 2015.

URL <http://dx.doi.org/10.1109/ISORC.2015.8>

During the design of embedded software, compilers play an important role, since the machine code generated by them directly influences criteria like, e.g., execution times, timing predictability or energy. Particularly, compiler optimizations could be beneficial to improve such criteria systematically.

The discussions during this seminar revealed that both the predictability and the security community lack suitable models and that, if models are available, they are often used in the form of black boxes. This presentation intends to show what can be done within a compiler when combining models that are usually used by different communities.

By coupling a compiler with a static timing analyzer, a formal WCET timing model based on micro-architectural features was integrated into the compilation flow. Next, this low-level hardware model is combined with a code-level control flow model that allows for the systematic optimization of WCETs by the compiler. Finally, task set-level models from the scheduling theory community are integrated into the optimization flow.

By means of a Scratchpad Memory (SPM) allocation, this presentation aims to show how complete multi-task sets can finally be optimized for timing predictability. Due to their timing predictability, SPMs are useful to achieve isolation between concurrent software tasks. By combining all these various models into the compiler’s optimization process, we are able to achieve predictability and inter-task isolation by controlling resource use statically at compile time for entire multi-task systems.

In the future, it would be worthwhile to investigate in how far the memory-related isolation achieved by our existing WCET-oriented optimizations are useful for security. Furthermore, a tight(er) connection between compilers and operating systems might be useful for more efficient and effective resource allocation decisions at runtime in order to finally achieve adaptive isolation.

References

- 1 Arno Luppold, Heiko Falk. *Code Optimization of Periodic Preemptive Hard Real-Time Multitasking Systems*. In Proceedings of the 18th International Symposium on Real-Time Distributed Computing (ISORC), Auckland / New Zealand, April 2015

5.4 Soteria: Offline Software Protection within Low-cost Embedded Devices

Johannes Götzfried (Universität Erlangen-Nürnberg, DE)

License © Creative Commons BY 3.0 Unported license
© Johannes Götzfried

Joint work of Johannes Götzfried, Tilo Müller, Ruan de Clercq, Pieter Maene, Felix C. Freiling, Ingrid Verbauwhede

Main reference J. Götzfried, T. Müller, R. de Clercq, P. Maene, F. C. Freiling, I. Verbauwhede, “Soteria: Offline Software Protection within Low-cost Embedded Devices”, in Proc. of the 31st Annual Computer Security Applications Conf. (ACSAC 2015), pp. 241–250, ACM, 2015.

URL <http://dx.doi.org/10.1145/2818000.2856129>

Protecting the intellectual property of software that is distributed to third-party devices which are not under full control of the software author is difficult to achieve on commodity hardware today. Modern techniques of reverse engineering such as static and dynamic program analysis with system privileges are increasingly powerful, and despite possibilities of encryption, software eventually needs to be processed in clear by the CPU. To anyhow be able to protect software on these devices, a small part of the hardware must be considered trusted. In the past, general purpose trusted computing bases added to desktop computers resulted in costly and rather heavyweight solutions. In contrast, we present Soteria, a lightweight solution for low-cost embedded systems. At its heart, Soteria is a program-counter based memory access control extension for the TI MSP430 microprocessor. Based on our open implementation of Soteria as an openMSP430 extension, and our FPGA-based evaluation, we show that the proposed solution has a minimal performance, size and cost overhead while effectively protecting the confidentiality and integrity of an application’s code against all kinds of software attacks including attacks from the system level.

5.5 Challenges of Temporal Isolation

Gernot Heiser (UNSW – Sydney, AU)

License © Creative Commons BY 3.0 Unported license
© Gernot Heiser

Joint work of Gernot Heiser, Anna Lyons, Thomas Sewell, Felix Kam, Qian Ge, Yuval Yarom

Main reference Q. Ge, Y. Yarom, G. Heiser, “Do Hardware Cache Flushing Operations Actually Meet Our Expectations?”, arXiv:1612.04474v3 [cs.CR], 2016.

URL <https://arxiv.org/abs/1612.04474v3>

Spatial isolation is well-supported by present hardware and software, e.g. the seL4 microkernel has been proved to support spatial isolation, including the absence of covert storage channels. While the formal arguments about seL4 presently only apply to a single-core version, the extension its functional verification to multicore hardware is in progress, and unlikely to produce issues in terms of spatial isolation.

In contrast, temporal isolation is not only harder to verify, hardware is becoming less predictable, thanks to an increasing number of performance-enhancement tricks, generally based on some form of caching and dynamic scheduling of resources. This makes it increasingly difficult, and in cases impossible, to bound and control non-determinism.

I argue that computer architects have essentially abandoned the instruction-set architecture (ISA) as the contract between hardware and software: by just referring to the ISA, it is impossible to guarantee safety (timeliness) and security (absence of timing channels).

I argue further that it is hopeless to address this problem unless architects agree to a usable contract, i.e. extend the ISA so that timing effects become visible (and thus analysable) or controllable.

In particular, there must be time bounds on all operations. In practice, bounding each individual operation (instruction) may not be enough, as this will lead to massively pessimistic bounds. As future hardware will never be fully utilisable (eg one cannot run all cores because they will overheat), this pessimism may be tolerable in many cases. In others, enough information must be available so that it is at least possible to obtain realistic bounds on the execution time of groups of operations, giving software the opportunity to re-introduce determinism at a higher level.

Examples of this are variations produced by shared state such as various forms of caches and interconnects, which produce variations in execution time that break isolation. Establishing safety requires the ability to bound variations. Establishing security is harder, as it requires establishing determinism, at least at some course granularity. This is possible as long as the hardware provides mechanisms to either partition or flush (with bounded latency) any such shared state.

5.6 Predictability in Multicore Systems Using Self-Suspending Tasks

Jian-Jia Chen (TU Dortmund, DE)

License © Creative Commons BY 3.0 Unported license
© Jian-Jia Chen

In general computing systems, a job (process/task) may suspend itself whilst it is waiting for some activity to complete. With the presence of self-suspension, the typical critical instant theorem cannot be directly applied. However, such suspending behavior is in general unavoidable unless the executions are isolated. In this talk, I present a short overview of typical schedulability tests, explain our observations why suspension is important to account for the impact of shared resources, and provide a brief overview of recent developments with regard to the schedulability tests.

5.7 Software Development for Isolation

Tulika Mitra (National University of Singapore, SG)

License © Creative Commons BY 3.0 Unported license
© Tulika Mitra

Joint work of Tulika Mitra, Konstantin Rubinov, Lucia Rosculete, Abhik Roychoudhury
Main reference K. Rubinov, L. Rosculete, T. Mitra, A. Roychoudhury, “Automated Partitioning of Android Applications for Trusted Execution Environments”, in Proc. of the 38th Int’l Conf. on Software Engineering (ICSE’16), pp. 923–934, ACM, 2016.
URL <http://dx.doi.org/10.1145/2884781.2884817>

The co-existence of critical and non-critical applications on computing devices is becoming commonplace. The sensitive segments of a critical application should be executed in isolation on Trusted Execution Environments (TEE) so that the associated code, data, and their execution can be protected from malicious applications both for security and timing predictability. TEE is supported by different technologies and platforms, such as ARM Trustzone, that allow logical separation of secure and normal worlds. However, software development on such platforms to take advantage of the hardware support for isolation remain challenging resulting in slow adoption of isolation techniques at application level. We develop an automated approach to help application developers adopt hardware-enforced secure technology for isolation by retrofitting original applications to protect sensitive data. Our approach automatically partitions critical Android applications into client code to be

run in the normal world and TEE code encapsulating the handling of confidential data to be run in the secure world. We further reduce the overhead due to transitions between the two worlds. The advantage of our proposed solution is evidenced by efficient automated partitioning of real-world Android applications to protect sensitive code/data.

5.8 Time-Based Intrusion Detection in Cyber-Physical Systems

Frank Mueller (North Carolina State University – Raleigh, US)

License © Creative Commons BY 3.0 Unported license
© Frank Mueller

Joint work of Christopher Zimmer, Balasubramanya Bhat, Frank Mueller, Sabin Mohan
Main reference C. Zimmer, B. Bhat, F. Mueller, S. Mohan, “Time-based intrusion detection in cyber-physical systems”, in Proceedings of the 1st ACM/IEEE Int’l Conf. on Cyber-Physical Systems (ICCPS’10), pp. 109–118, ACM, 2010.
URL <http://dx.doi.org/10.1145/1795194.1795210>

Security in real-time cyber-physical systems (CPS) has been an afterthought, even though such systems are networked. We present three mechanisms for time-based intrusion detection exploiting information obtained by static timing analysis. For real-time CPS systems, timing bounds on code sections are readily available as they are calculated during schedulability analysis. We demonstrate how checks of micro-timings at multiple granularities of code uncover intrusions (1) in a self-checking manner by the application and (2) through the operating system scheduler, which has never been done before.

5.9 Adaptive Pipeline for Security in Real Time Systems

Sri Parameswaran (UNSW – Sydney, AU)

License © Creative Commons BY 3.0 Unported license
© Sri Parameswaran

Joint work of Amin Malekpour, Sri Parameswaran, Roshan Ragel

Hardware Trojans are employed by adversaries to either leak information or to prevent computation deliberately by inserting alterations at design time. Hardware Trojans compromise the operation of systems, reducing the trust placed in any manufactured hardware, as well as any software executing upon that hardware. A Trojan can be always ON or be triggered by a certain condition either external or internal. Even before the manufacturing process, intellectual property (3PIPs) cores supplied by third-party vendors as well as electronic design automation (EDA) tools (developed by various companies) could well make the in-house design process of ICs vulnerable. During the typical development cycle of an IC, each party associated with design, manufacturing and distribution of an IC can be a potential adversary, who could well insert undesired malicious modifications into the IC. Therefore, either ensuring that the ICs are free of hardware Trojans or mitigating their harmful impact is important. Most existing countermeasures focus on the difficult task of detecting and preventing hardware Trojans. Although Trojan identification before ICs are deployed in the system can be beneficial, the proposed techniques for detection cannot guarantee detection of all types and sizes of Trojans. We aim to apply online monitoring notion to a Pipelined Multiprocessor System-on-Chip (PMPSoC), which enables the system to work safely in the presence of Trojans while utilizing shelf commercial processing elements (3PIPs). The system adapts and randomizes to provide security. Our proposed online monitoring would facilitate the detection/recovery of/from hardware Trojan attacks, albeit with some overheads. Our system is implemented as PMPSoC architecture and uses a diverse set of 3PIPs.

5.10 Timing Predictability and How to Achieve It

Jan Reineke (*Universität des Saarlandes, DE*)

License © Creative Commons BY 3.0 Unported license
© Jan Reineke

Main reference S. Hahn, J. Reineke, R. Wilhelm, “Toward Compact Abstractions for Processor Pipelines”, in Proc. of the Correct System Design Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, LNCS, Vol. 9360, pp. 205–220, Springer, 2015.

URL http://dx.doi.org/10.1007/978-3-319-23506-6_14

For hard real-time systems, timeliness of operations has to be guaranteed. Static timing analysis is therefore employed to compute upper bounds on the execution times of a program. Analysis results at high precision are required to avoid over-provisioning of resources.

In the first part of the talk, I stress the need for faithful microarchitectural models. Without such models no reliable predictions about a program’s future execution times can be made. Unfortunately, models at the level of detailed required for timing analysis are rarely available.

In the second part of the talk, I discuss the notions of timing predictability and analyzability. Timing predictability is related to the range of possible execution times of a program under different conditions, such as different initial hardware states or different amounts of interference generated by co-running tasks on other processor cores. Modern microarchitectural features such as deep pipelines, complex memory hierarchies, and shared resources in multi or many cores generally decrease predictability.

I discuss three approaches to increase predictability:

1. Eliminating stateful components: e.g. by replacing caches by scratchpad memories, or an out-of-order architecture by a VLIW architecture. The challenge then is the efficient static allocation of resources.
2. Eliminating interference: this is achieved by partitioning shared resources in time and/or space. The challenge is the efficient partitioning of the resources.
3. Choosing “forgetful” components, i.e., components whose behavior is relatively insensitive to its initial state. For caches we know that LRU replacement in this regard. For other microarchitectural components, our understanding is less developed.

Timing analyzability is concerned with analysis efficiency. Analyzability can be improved by the same three approaches that increase predictability:

1. Eliminating stateful resources results in fewer hardware states that timing analysis needs to account for.
2. Eliminating interference allows timing analysis to safely ignore the behavior of co-running tasks.
3. Different initial states will quickly converge during analysis for forgetful components.


While the three approaches discussed above are beneficial for both predictability and analyzability, two properties of timing models are related primarily to analyzability:

1. Freedom from timing anomalies, and
2. Timing compositionality

Both properties enable the implicit and thus efficient treatment of large sets of hardware states during timing analysis. I show that even models of simple in-order processors are neither free from timing anomalies nor timing compositional. Finally, I sketch “strictly in-order pipelines”, which are provably free from timing anomalies and timing compositional.

5.11 Connecting the dots – Towards the total automation of embedded systems design (in Java world)

Zoran Salcic (University of Auckland, NZ)

License  Creative Commons BY 3.0 Unported license
© Zoran Salcic

Java, although used in large number of embedded systems, still has not found its proper place in research community. By extending Java with GALS abstractions and formal model of concurrency and reactivity, we give it a new life. SystemJ language, initially aimed at general concurrent and distributed systems has found its way to embedded real-time world to become a language with which a design begins and goes through various transformations until it finds a suitable/customised multicore platform for its execution. We will talk about how the dots are connected, the central role of a formal representation of SystemJ program in all phases and variations of design process. Multi-dimensional research has been developed related to generation of efficient code that runs on a time-predictable multi-core platform, satisfies timing constraints proven via static analysis and allows generation of the execution platform suitable for further requirements such as fault-tolerance, isolation of software behaviours using spatial and temporal methods within the platform's NoC, resource-aware program execution etc.

5.12 Isolation for Security

Patrick Schaumont (Virginia Polytechnic Institute – Blacksburg, US)

License  Creative Commons BY 3.0 Unported license
© Patrick Schaumont

Modern information infrastructure is very heterogeneous, with the Internet of Things on the one end, and the Cloud on the other. Computing capabilities, storage, and computing performance vary by orders of magnitude as one moves from the IoT to the cloud. I used the example of implantable/wearable medical devices to illustrate this point, and to address the security concerns that occur within such information infrastructure [1, 2]. In particular, there are requirements for information security, safety, and privacy. These requirements affect the entire information chain from implanted device up to the cloud server.

Hence, when considering 'isolation for security', it is vital to do this within the proper architectural context. Second, the architecture has significant impact on the implementation of security. Constrained implementations, found near the outer periphery in the internet of things, use simple micro-controllers, simple cryptography, and static secrets. High-end implementations, found in the cloud, use advanced processors, complex public-key crypto, and ephemeral secret. Moreover, many of the high-end architectures have isolation naturally build-in.

A central concept in the design of secure architectures is that of trust. A computer system is trusted when it behaves as expected. A computer system that is not trusted has unknown behavior – that is, it may work as we expect it should, or it may not. We just don't know. The boundary between the trusted part and the non-trusted part of a computer system is the trust boundary.

A closely related concept is that of the attacker model [3]. An attacker model enumerates the mechanisms through which the adversary will try to breach the trust boundary. In computer systems, this can be done in various ways. The I/O attacker model assumes an

attacker who can manipulate the input of a system in order to cause an internal exception and take control. The Machine code attacker model assumes an attacker who coexists in the same memory space as a secure task, thereby introducing the requirement to enforce strict isolation between the secure task and the rest of the system. The Hardware attacker model represents the strongest attack and assumes an attacker who has (to some extent) physical control over the computer architecture.

During the talk, I discussed two examples of computer systems that are able to handle the Machine code attacker model, including SGX from Intel [5, 6] and SANCUS, developed at KU Leuven [4].

Some final open issues are (a) how security can be quantified; (b) what metrics would be suitable to describe secure computing and (c) what are the orthogonal properties of secure computing (next to isolation).

References

- 1 Michael Rushanan, Aviel D. Rubin, Denis Foo Kune, Colleen M. Swanson: *SoK: Security and Privacy in Implantable Medical Devices and Body Area Networks*. IEEE Symposium on Security and Privacy 2014: 524–539.
- 2 Wayne Burleson, Shane S. Clark, Benjamin Ransford, Kevin Fu: *Design challenges for secure implantable medical devices*. DAC 2012: 12–17.
- 3 Frank Piessens, Ingrid Verbauwhede: *Software security: Vulnerabilities and countermeasures for two attacker models*. DATE 2016: 990–999.
- 4 Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, Frank Piessens: *Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base*. USENIX Security Symposium 2013: 479–494.
- 5 Victor Costan, Srinivas Devadas: *Intel SGX Explained*. IACR Cryptology ePrint Archive 2016: 86 (2016).
- 6 Ittai Anati, Shay Gueron, Simon Johnson, Vincent Scarlata: *Innovative Technology for CPU Based Attestation and Sealing*. Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP 2013.

5.13 T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems

Martin Schoeberl (Technical University of Denmark – Lyngby, DK)

License © Creative Commons BY 3.0 Unported license
© Martin Schoeberl

Main reference M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, A. Tocchi, “T-CREST: Time-predictable multi-core architecture for embedded systems”, *Journal of Systems Architecture*, Vol. 61(9), pp. 449–471, Springer, 2015.

URL <http://dx.doi.org/10.1016/j.sysarc.2015.04.002>

Real-time systems need time-predictable platforms to allow static analysis of the worst-case execution time (WCET). Standard multi-core processors are optimized for the average case and are hardly analyzable. Within the T-CREST project we propose novel solutions for time-predictable multi-core architectures that are optimized for the WCET instead of the average-case execution time. The resulting time-predictable resources (processors, interconnect, memory arbiter, and memory controller) and tools (compiler, WCET analysis)

are designed to ease WCET analysis and to optimize WCET performance. Compared to other processors the WCET performance is outstanding.

The T-CREST project is the result of a collaborative research and development project executed by eight partners from academia and industry. The European Commission funded T-CREST.

5.14 Adaptive Memory Protection for Many-Core Systems

Wolfgang Schröder-Preikschat (Universität Erlangen-Nürnberg, DE)

License © Creative Commons BY 3.0 Unported license
© Wolfgang Schröder-Preikschat

Joint work of Gabor Drescher, Wolfgang Schröder-Preikschat

Memory protection based on MMUs or MPUs is widely applied in all areas of computing from mobile devices to HPC. It is a basic building block to provide security between the OS kernel and applications but also among different applications. However, unprotected execution is also generally possible and typically exerted in resource-restricted environments, for instance embedded systems. Both variants have their advantages and disadvantages. While memory protection ensures safety and security in the face of arbitrary programs, it is also costly to manage. Updates of multi-level page-table data structures, TLB invalidations via inter processor interrupts and page faults on memory accesses are significant sources of unpredictability.

State of the art operating systems statically determine which applications or application's modules run under memory protection and which do not. This assignment of protection does not change at run time. This means, software is either restricted to solely access its own memory regions and this is enforced by hardware mechanisms, or it may access all memory regions freely.

There are scenarios where ubiquitous memory protection is unnecessary and increases uncertainty for some time, but is required at other points during the run time of a system. The simplest example is the execution of a single application, kernel regions need to be protected but otherwise the application may freely access all available memory without error. Once another application is started, both may need confinement. In the case of applications written in a type-safe language, memory-protection overheads are also wasteful, as the application cannot perform arbitrary pointer arithmetic. Another scenario may be real-time applications where time predictability may be of higher interest than security. Finally, applications of the same vendor may trust each other but mistrust software of different origin. Memory protection may be superfluous in this scenario until foreign software is started on the system.

This talk presents an adaptive memory-protection system that is capable of dynamically changing the protection state of applications from protected to unprotected and back again. This adaptability applies at run time to parallel applications utilizing dynamic memory allocation. It allows the combination of the benefits of both worlds: security when memory protection is needed and increased performance and predictability once security is superfluous. Evaluation results for up to 64 cores on a contemporary x86 64 bit server show reduced time complexity of relevant system services from linear to constant time in the unprotected case. Unprotected applications benefit from faster system calls, starting at a 3.5 times speedup. Furthermore, it can be shown that unpredictable running times and system call latencies can be reduced. Nevertheless, the OS maintains the ability to confine applications at any time.

5.15 Security Issues on the Boundary of Ideal and Real Worlds

Takeshi Sugawara (Mitsubishi – Kanagawa, JP)

- License** © Creative Commons BY 3.0 Unported license
© Takeshi Sugawara
- Joint work of** Takeshi Sugawara, Daisuke Suzuki, Minoru Saeki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki, Takeshi Fujino
- Main reference** T. Sugawara, D. Suzuki, R. Fujii, S. Tawa, R. Hori, M. Shiozaki, T. Fujino, “Reversing Stealthy Dopant-Level Circuits”, in Proc. of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2014), Journal of Cryptographic Engineering, Vol. 5(2), pp. 85–94, Springer, 2015.
URL <http://dx.doi.org/10.1007/s13389-015-0102-5>
- Main reference** T. Sugawara, D. Suzuki, M. Saeki, M. Shiozaki, T. Fujino, “On Measurable Side-Channel Leaks Inside ASIC Design Primitives”, in Proc. of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2013), Journal of Cryptographic Engineering, Vol. 4(1), pp. 59–73, Springer, 2014.
URL <http://dx.doi.org/10.1007/s13389-014-0078-6>

Assumption (i.e., model, abstraction) is essential in security because it is the interface between theorists and experimentalists. Good assumption can be easily verified by experimentalists. In the talk, my recent results on the research of side-channel attack and LSI reverse engineering are briefly introduced in order to show examples how the countermeasures are constructed and how their assumptions are falsified. Finally, static isolation based on domain-specific co-processors, that is being common in industry is explained.

5.16 An Introduction to the Seminar

Jürgen Teich (Universität Erlangen-Nürnberg, DE)

- License** © Creative Commons BY 3.0 Unported license
© Jürgen Teich

Presented is an introduction and motivation of the topic of this Dagstuhl Seminar: Adaptive isolation of applications on Multi-Core Systems in order to provide, improve or enforce timeliness of computations as well as security on demand.

First, different definitions on timing predictability are revisited and restriction of input spaces as well as isolation of resources discussed for in improving timing predictability or allowing analyzability at all. Subsequently, major sources of unpredictability are summarized, including sharing of resources, multi-threading, and power management techniques as used today.

For isolation, resource reservation protocols, virtualization techniques and invasive computing, a new paradigm for parallel multi-core computing in which applications “invade” resources on demand in order to restrict the interference with other applications is introduced.

It is shown that for many applications such as stream processing, a formal timing analysis is possible. Also, the variation of execution time may be greatly reduced through the isolation created by invading isolated rather than sharing. Finally, invasive computing may also be used to virtualize a multi-core platform and provide secure islands on demand.

Conclusions are given to point out similarities and differences between isolation techniques for time predictability and properties important in the domain of security.

5.17 Isolation, resource efficiency and covert channels

Lothar Thiele (ETH Zürich, CH)

License © Creative Commons BY 3.0 Unported license
© Lothar Thiele

Joint work of Lothar Thiele, Miedl Philipp

Modern multicore processors feature easily accessible temperature sensors that provide useful information for dynamic thermal management. These sensors were recently shown to be a potential security threat, since otherwise isolated applications can exploit them to establish a thermal covert channel and leak restricted information. Previous research showed experiments that document the feasibility of (lowrate) communication over this channel, but did not further analyze its fundamental characteristics. For this reason, the important questions of quantifying the channel capacity and achievable rates remain unanswered.

To address these questions, we devise and exploit a new methodology that leverages both theoretical results from information theory and experimental data to study these thermal covert channels on modern multicores. We use spectral techniques to analyze data from two representative platforms and estimate the capacity of the channels from a source application to temperature sensors on the same or different cores. We estimate the capacity to be in the order of 300 bits per second (bps) for the same-core channel, i.e., when reading the temperature on the same core where the source application runs, and in the order of 50 bps for the 1-hop channel, i.e., when reading the temperature of the core physically next to the one where the source application runs. Moreover, we show a communication scheme that achieves rates of more than 45 bps on the same-core channel and more than 5 bps on the 1-hop channel, with less than 1% error probability. The highest rate shown in previous work was 1.33 bps on the 1-hop channel with 11% error probability.

5.18 Determinate and Timing-Predictable Concurrency in Reactive Systems – The Synchronous Approach and the SCCharts Language

Reinhard von Hanxleden (Universität Kiel, DE)

License © Creative Commons BY 3.0 Unported license
© Reinhard von Hanxleden

Joint work of Joaquin Aguado, David Broman, Björn Duderstadt, Insa Fuhrmann, Reinhard von Hanxleden, Michael Mendler, Steven Loftus-Mercer, Christian Motika, Owen O’Brien, Partha Roop, Steven Smyth, Alexander Schulz-Rosengarten, KIELER and ELK teams

Main reference R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, O. O’Brien, “SCCharts: sequentially constructive statecharts for safety-critical applications: HW/SW-synthesis for a conservative extension of synchronous statecharts”, in Proc. of the 35th ACM SIGPLAN Conf. on Prog. Lang. Design and Implementation (PLDI’14), pp. 372–383, ACM, 2014.

URL <http://dx.doi.org/10.1145/2594291.2594310>

Synchronous programming languages are well established for programming safety-critical reactive systems that demand determinate behavior and predictable timing. One commercially successful example is the graphical modeling language provided by the Safety Critical Application Development Environment (SCADE), which is used for e.g. flight control design. Another, more recently developed synchronous language are SCCharts, a statechart variant that extends classical synchronous programming with a more liberal, but yet determinate scheduling regime for shared variables. SCCharts can be compiled into both software and hardware, with a sequence of structural model-to-model transformations that allow to map the temporal behavior back to the model. The presentation emphasizes that the compilation approach has a high influence on timing predictability.

5.19 Hybrid Application Mapping for Dynamic Isolation in Invasive Computing

Stefan Wildermann (Universität Erlangen-Nürnberg, DE)

License © Creative Commons BY 3.0 Unported license
© Stefan Wildermann

Joint work of Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, Jürgen Teich
Main reference A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Glaß, J. Teich, “DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems”, in Proc. of the Int’l Conf. on Hardware/Software Codesign and System Synthesis (CODES’14), pp. 34:1–34:10, ACM, 2014.

URL <http://dx.doi.org/10.1145/2656075.2656083>

Multi-Processor Systems-on-a-Chip (MPSoCs) provide sufficient computing power for many applications in scientific as well as embedded applications. Unfortunately, when real-time, reliability, and security requirements need to be guaranteed, applications suffer from the interference with other applications, uncertainty of dynamic workload and state of the hardware. Composable application/architecture design and timing analysis is therefore a must for guaranteeing applications to satisfy their non-functional requirements independent from dynamic workload.

Invasive Computing can be used as the key enabler, as it provides the required isolation of resources allocated to each application. On the basis of this paradigm, this work presents a hybrid application mapping methodology that combines design-time analysis of application mappings with run-time management. Design space exploration delivers several resource reservation configurations with verified and/or validated non-functional properties of individual applications. These properties can then be guaranteed at run-time, as long as dynamic resource allocations comply with the offline analyzed resource configurations. In this work, we show that the approach provides increased flexibility and dynamism of systems even in the presence of hard real-time constraints. We also show the overhead of performing this dynamic application isolation based on run-time resource allocation.

5.20 Timing Verification – Flogging a Dead Horse?

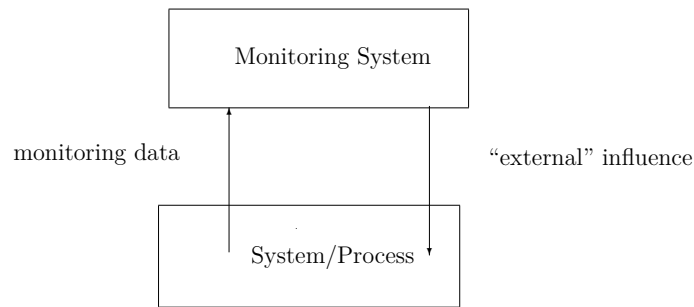
Reinhard Wilhelm (Universität des Saarlandes, DE)

License © Creative Commons BY 3.0 Unported license
© Reinhard Wilhelm

Joint work of Jan Reineke, Sebastian Hahn, Reinhard Wilhelm
Main reference P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. von Hanxleden, R. Wilhelm, W. Yi, “Building timing predictable embedded systems”, ACM Trans. Embed. Comput. Syst., Vol. 13(4), pp. 82:1–82:37, ACM, 2014.

URL <http://dx.doi.org/10.1145/2560033>

Recent developments in architectures and their adoption for safety- and time-critical embedded systems have reduced or even eliminated the chance to apply sound timing-verification techniques. The complexity, i.e., the size of the state space to be explored, has become just too large. The deployed new architectures have even made measurement-based, i.e. unsound techniques unpractical. What remains as alternative? The participants of this Dagstuhl Seminar have found no answer to this question. Maybe the timing-verification community should provide the proof that high-performance, predictable architectures are possible.



■ **Figure 1** Conceptual model of a runtime monitoring system.

6 Working groups

6.1 Runtime Monitoring

Felix Freiling (Universität Erlangen-Nürnberg, DE)

License Creative Commons BY 3.0 Unported license
© Felix Freiling

This breakout session ran for two days and focussed on the general topic of runtime monitoring for predictability and security. The participants on Wednesday were Lothar Thiele, Pieter Maene, Johannes Götzfried, Takeshi Sugawara, Jürgen Teich and Felix Freiling. On Thursday, the participants were Jan Reineke, Pieter Maene, Johannes Götzfried, Takeshi Sugawara, Jürgen Teich, Felix Freiling and Sudipta Chattopadhyay.

6.1.1 Conceptual model

We started with a conceptual model to clarify what we were talking about (see Figure 1). We usually have two systems: On the one hand there is the monitored system, which is the system whose properties we wish to monitor. If the monitored system would constantly satisfy its desired properties, there would be no necessity to monitor it. So we have the second system: the monitoring system. The monitoring system collects data of the monitored system (either digital or physical, e.g. using a camera) and has some internal logic that performs computation on this data. Under certain conditions, the monitoring system might influence the monitored system, e.g. by resetting the system or initiating a reconfiguration.

Formally, the observations of the monitoring system can be viewed as a sequence of events or states observed from the monitored system. Such sequences are often called *traces*. In a security context, the trustworthiness of the observed data is an issue. But also in a timeliness context the precision of the observed timings is critical to make truthful/good decisions. The classical examples of runtime monitoring systems are:

- watchdog timers,
- fault-injection attack countermeasures using sensors [2], or
- intrusion detection services within networks.

6.1.2 Issues in runtime monitoring

The discussion identified several issues that we used to structure the area of runtime monitoring systems. We discuss each of them separately in the following sections.

6.1.2.1 Types of properties that are monitored

We collected a set of properties that are on the usual wishlist of runtime monitoring systems:

- timeliness (i.e. meeting realtime deadlines),
- control flow integrity (i.e. the control flow does not deviate from the “correct” control flow intended by the programmer), or
- variance of response times (i.e., the variance of response times within the last hour is below a certain value).

It was noted that in general, properties that can be detected on individual traces fall in the class of safety properties (in the safety/liveness sense of Lamport [6] and Alpern and Schneider [1]). It is theoretically possible to derive bad events (or event sequences) from safety properties and to configure detection conditions for them.

It is well-known that certain types of functional and non-functional properties do not fall into the class of safety properties. For example, liveness properties (e.g., eventual termination) cannot be detected at runtime since they are only satisfied in infinite time (to detect violations one would have to observe the system infinitely long). But also variance of response times or information flow cannot be modeled as trace sets, but are a property of all traces of a system (see McLean [7]). They therefore cannot be checked at runtime. However, many such properties can be approximated by safety properties detectable at runtime (see work by Schneider [8]). For example, variance can be approximated using a finite time window in the past, or information flow can be approximated by also using the amount of entropy of the state trace of certain resources in the past.

Sometimes, it may be necessary or feasible to not detect precise events but rather observe “anomalies” of some form. This is the usual approach in intrusion detection where it is sometimes not clear how the attacker will attack and leave traces. If it is not totally clear what to detect, there are additional problems. For example, an attacker could try to avoid detection by trying to hide “beneath the radar”, e.g., by performing changes slowly enough so that they do not cause an anomaly.

6.1.2.2 Monitoring approaches

There can be different types of monitoring approaches. Which one is selected depends on the type of property being enforced and the type of fault/adversary/attacker being considered.

There is the distinction between permanent and periodic monitoring. Monitoring can also be done on demand or periodically but not in fixed intervals but in random (unpredictable) intervals. Given an intelligent adversary, periodic monitoring can be avoided because the attacker can wait for a check, then perform the attack and cover all traces before the next check occurs (sometimes called “ABA problem”).

In the literature, there is also the notion of a passive monitoring approach known as canaries. The idea is to detect fault using an artifact that is sensitive and easily broken by the fault:

- software canary (also known as cookie) which is used for buffer overflow protection like in StackGuard [4],
- active shield used to detect invasive probing of a circuit [3], and
- a special data store used to detect timing violation (i.e., canary flip flop).

6.1.2.3 Interactions between monitoring and monitored system

Usually it is assumed that monitoring and monitored system are subject to different types of faults/attacks. In security, this is realized through privilege separation (user/system level,

processor rings, ARM Trustzone, Intel SGX, etc.), such that the attacker assumption can be justified that the privileged area is not affected by the attack. Without such a restriction on the attacker no security can be guaranteed (see breakout discussions on “attacker models”).

In the safety/realtime/fault-tolerance area it is sometimes the case that two systems take on the role of monitoring and monitored system for each other (see for example self-checking checkers from coding theory or fault-tolerant protocols for voting or agreement). In such scenarios the fault assumption is restricted in the way that both monitoring and monitored system can be affected by faults but not both at the same time. In this context, we often find fault/attacker assumptions such as “ k -out-of- n ” (meaning that at most k out of n systems are affected by faults).

6.1.2.4 Suitable reactions

In case the monitoring system issues a reaction, what can this be? In fault-tolerant systems there exists the notion of fail-safe meaning that the system is switched from operational to a safe state (e.g. “all signals stop” in railway systems). This is problematic in situations where there is no safe alternative to normal operation (as in avionics). In this case, the minimum you can do is to at least ensure that sufficient evidence is collected for a later (post mortem) analysis, i.e. work which has been done under the heading of secure logging.

In security there is the notion of fail-secure meaning that if the system fails, security properties like integrity or confidentiality are not violated or re-established (e.g., forward secrecy). Interestingly, availability is not very often investigated in a security context. Availability, however, is important in this context since reactions to faults/attacks usually mean a form of reset or restart, and continuous triggers to restart can cause unavailability (denial-of-service attacks). This is especially problematic when unavailability is the result of false detections (false positives) of the monitoring system.

A suitable reaction usually is to adjust resources necessary for the task and continue the task with better resources. Continuation can mean that it restarts from a previous (uncorrupted) checkpoint. In this context, the notion of stabilization was mentioned [5] meaning that as long as faults/attacks happen, no progress is guaranteed, but that the system automatically regains progress once faults/attacks stop to occur.

6.1.3 Open problems

While most of the above observations can be considered known or at least named in the literature, we collected a couple of open points that were considered novel aspects of the problem that deserved some research attention:

- The whole issue of *distributed monitoring* has its problems of its own: Distribution creates challenges with the attacker model, trust issues in the exchange of information, problems of global observation etc.
- If a monitoring system is needed, why not generate it automatically? Given a monitoring property, can we automatically generate a monitoring system in software and/or hardware that observes it?
- What is the correct/right granularity of checking the monitored property? Should monitoring be done periodically, on demand or continuously? What is the relation to efficiency of the monitoring process?
- Monitoring usually assumes that there is some information against which a monitored property can be checked? Can this signature be normalized? Can it possibly be reduced to the knowledge of a (cryptographic) key?

- Adding monitoring functionality increases the attack surface of the program. To what extent does the monitoring functionality affect security then?
- In what sense are safety properties also security properties? Obviously this depends on the attacker model: Which attacker model comprises which fault model? In case an attacker model includes a fault model, in what way does runtime monitoring for faults subsume monitoring effort for attacks? These questions refer to the synergies of faults vs. attack detection.

References

- 1 B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
- 2 Josep Balasch. Introduction to fault attacks. Presentation at IACR Summer School, Chia Laguna, Sardinia, October 2015. https://www.cosic.esat.kuleuven.be/summer_school_sardinia_2015/slides/Balasch.pdf.
- 3 Sébastien Briaïs, Stéphane Caron, Jean-Michel Cioranescu, Jean-Luc Danger, Sylvain Guilley, Jacques-Henri Jourdan, Arthur Milchior, David Naccache, and Thibault Porteboeuf. 3D hardware canaries. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012 – 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2012.
- 4 Crispan Cowan. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In Aviel D. Rubin, editor, *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*. USENIX Association, 1998.
- 5 E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- 6 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, March 1977.
- 7 John McLean. A general theory of composition for a class of “possibilistic” properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, January 1996.
- 8 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, February 2000.

6.2 Future of Timing Verification

Samarjit Chakraborty (TU München, DE)

License © Creative Commons BY 3.0 Unported license
© Samarjit Chakraborty

The breakout session explored the future of timing verification of real-time systems. This breakout session was motivated by the fact safety-critical embedded systems are increasingly relying on advanced processor architectures that have been designed with the goal of improving average-case performance and not predictability. Hence, while there have been considerable advancements in the domains of Worst Case Execution Time (WCET) analysis of programs, and also timing analysis of real-time systems, this is increasingly appearing to be a losing battle. More importantly, academic research in the domains of WCET and timing analysis has had little impact on practice. Therefore, the question is what is the path forward from here?

Here, one line of thought that has emerged during the last 1-2 years, especially in the context of embedded control systems is how many deadlines and which ones should really

be met in a real-time system? In other words, is 100% predictability, as aimed in real-time systems research really needed? Typically, many safety critical systems implement some control algorithm. Meeting control performance requirements are subject to satisfying some timing constraints, which a real-time systems theorist tries to verify (schedulability analysis) or ensure (schedule design or synthesis). Hence, deadline constraints have served as a good interface between control theorists and real-time/embedded systems theorists.

However, most feedback control systems have an inherent degree of robustness. Also when the plant is in a “steady” state, the system can be run in open loop (i.e., no computation of control input is necessary). This means that even if some control signals are not computed or transmitted in time, the control performance still remains acceptable. If these control signals can be characterized, i.e., acceptable patterns of deadline misses may be computed then they might be interpreted as a quantification of the degree of predictability that is needed.

This means that timing analysis, instead of focusing on deadline constraints, should focus on higher-level (control theoretic) goals that better characterize the systems performance requirements. However, computing acceptable patterns of deadline violations is not trivial and requires sophisticated control theoretic analysis. For example, see [1, 2]. Further, timing or schedulability analysis to ascertain that at most these deadline violations may happen is more difficult than checking that no deadline violations happen. Nevertheless, such an approach gives a certain leeway that could be exploited to allow platforms and architectures that cannot be completely analyzed but instead only certain timing bounds on their performance may be given.

The discussion during this breakout session was also on the need for end-to-end timing analysis, e.g., considering the influence of operating systems on the execution time of code, which has not been sufficiently addressed until now. Finally, the need for benchmarks and the reproducibility of timing analysis techniques were also discussed. One potential solution would be to consider Simulink models of different controllers (e.g., from the automotive domain) and use the code synthesized from these models for timing analysis.

References

- 1 Dip Goswami, Reinhard Schneider, Samarjit Chakraborty. Relaxing Signal Delay Constraints in Distributed Embedded Controllers. *IEEE Trans. Control Systems Technology* 22(6): 2337-2345, 2014
- 2 Dip Goswami, Samarjit Chakraborty, Purandar Bhaduri, Sanjoy K. Mitter. Characterizing feedback signal drop patterns in formal verification of networked control systems. *IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, 2013

6.3 Attack Models

Albert Cohen (ENS – Paris, FR) and Karine Heydemann (UPMC – Paris, FR)

License © Creative Commons BY 3.0 Unported license
© Albert Cohen and Karine Heydemann

This breakout session focused on a general survey of attack models. The participants were Albert Cohen, Ruan De Clercq, Felix Freiling, Gernot Heiser, Karine Heydemann, Patrick Koeberl, Peter Maene, Claire Maiza, Sibin Mohan, Frank Mueller, Patrick Schaumont, and Takeshi Sugawara.

System designers need to define security properties and protective measures to implement them. This process involves systematic characterization of attack models. The working group

conducted survey of logical and physical attacks targeting CPS and IoT devices. The focus was on threats and attack models in general, in those associated with multi- and many-core systems in particular. Building on such a survey, researchers will be able to determine and to quantify the aspects of the design and implementation methods that need to be revisited, to integrate the security dimension at the heart of defense in depth mechanisms, correct-by-construction design, test, verification, validation, and certification.

6.4 Synergy between Predictability and Security

Frank Mueller (North Carolina State University – Raleigh, US)

License  Creative Commons BY 3.0 Unported license
© Frank Mueller

This break-out session was attended by about 25 participants and included other break-out ideas on availability, multi-cores, and levels on security.

In an initial brain-storming session, the full breadth of the topic was covered, where each participant contributed their ideas. The second session was dedicated to intensive discussions on various topics and concluded by various action items, including a security attack contest using predictability as a means of attack. The brain-storming ideas and following discussions are summarized under a number of topic headings:

6.4.1 Predictability versus Security

Participants observed that some timing predictability techniques are synergistic with security (and vice versa) while others are antagonistic in the sense that they appear to be in direct conflict. Synergistic examples range from timing information already available due to real-time analysis used for intrusion detection over obfuscation techniques in scheduling to the simplicity of crypto-algorithms facilitating their timing predictability. Antagonistic examples range from real-time predictability that may facilitate attacks over timing faults as a means to attack systems to a discussion on whether or not randomized attacks completely void any attempts to increase software diversity (including but not limited to parallelism/scheduling) when the number of variants is fixed.

6.4.2 Parallelism

One discussion was dedicated to the challenges of parallelism with diverse opinions. While isolation (in space, e.g., via partitioning, or in time, e.g., via TDMA) helps both predictability and security most of the time, it may come at a certain cost. For example, MMU protection results in page faults, which are costly, yet certain data structures may not require stringent access protection via MMUs (or MCUs), i.e., the idea of different levels of security seems intriguing.

6.4.3 Side-channel Attacks

The discussion centered around software-based side-channel attacks. Different types of leaked information (especially with respect to timing) and counter-measures were discussed. Mitigation techniques, e.g., obfuscation via randomization, were noted to adversely affect predictability while isolation typically aids predictability.

6.4.4 Availability

It was noted that no viable solution for predictability appears to exist. Once compromised, a hardware unit can always be powered down, even without physical access (assuming firmware safe-guards can be circumvented or are also affected by the intrusion). The idea of an analogy to priority inversion in security of a lower critical task affecting a higher critical one was discussed. Containment methods to handle faults by switching to more simplistic, higher protection modes also seem attractive in this context. But more fundamental work is required to better understand this subject.

6.4.5 Connectivity

Not much time was spent on discussing connectivity modes (on/off), but it was noted that isolation in proprietary networks can help. Independently, a need for protecting edge devices (e.g., IoT) was voiced as they, in large numbers, can easily orchestrate a DDOS attack.

6.4.6 Wish List

We formulated a wish list of action items. One challenge is to come up with a 3-dimensional model that combines security, predictability and also safety/fault tolerance as all three are inter-connected. Another challenge is the need for a hierarchy/levels, possibly for each of these three areas, and/or in a combined model. Different degrees of protection, degrees of information leakage, affected software mechanisms etc. may require a different response in protection. But most of all, security should be a first-order design principle, not an after-thought, as it currently is. And while fundamental security research is still required that may lead to completely new design methods, research is nonetheless needed to devise methods for retrofitting security into existing hardware/software systems.

6.4.7 The Programming Challenge

A final discussion culminated in the idea of posing a programming challenge for a successful timing attack, possibly in an Autosar setting. If successful, extensive PR should be used to put timing problems into the limelight within the realm of security, much in line with the way that the real-time system community received credit for saving the Mars Lander mission due to priority inheritance support on its software platform. One open question was if a follow-on 1-week attack hackathon should be organized to achieve this goal.

6.5 Models of Computation and Programming Languages

Reinhard von Hanxleden (Universität Kiel, DE)

License  Creative Commons BY 3.0 Unported license
© Reinhard von Hanxleden

There exist a multitude of models of computation (MoCs) and associated programming languages. Clearly, there is not a single winner, each has its pros and cons, and often a combination of languages is used in different parts of the design and at different abstraction levels. The aim of this break out session, based on two proposals brought in by David Broman and Reinhard von Hanxleden, was to identify the specific issues that arise for adaptive isolation. A number of questions were identified in advance, such as which the

essential language constructs are, which concepts should be left out, and what can we learn from currently available real-time languages and APIs. Given the setting of the seminar on multiprocessor systems on chip (MPSoCs), the role of concurrency was also brought forward as a possible focus. The participants in the breakout session were Davide Bertozzi (Università di Ferrara, IT), David Broman (KTH Stockholm, SE), Reinhard von Hanxleden (U Kiel, DE), Frank Mueller (North Carolina State University – Raleigh, US), Zoran Salcic (University of Auckland, NZ), Martin Schoeberl (Technical University of Denmark – Lyngby, DK), Stefan Wildermann (Universität Erlangen-Nürnberg, DE) and the aforementioned proposers.

As it turned out, the session participants were mostly from the predictability field, thus the resulting discussions centered around that aspect and had little focus on security. Furthermore, rather than trying to propose specific MoCs and languages, the participants agreed to try to identify properties that suitable MoCs and languages should have (even though specific MoCs/languages were covered to some extent).

To start with, possible application areas were discussed, including industrial control, image processing such as in a lane following assistant, and generally cyber-physical systems. There was a question mark on whether also financial applications would be within the scope, but later it was agreed that these would possess similar qualities.

Next, it was discussed what we try to avoid. There are obviously non-desirable scenarios, such as catastrophic failures in power plants and the like, but also “blue screens” that indicate undesired system states and loss of functionality and consequently may lead to a bad product reputation and resulting economic consequences. On a more technical level, a MoC/language should try to rule out things like memory leaks, timing issues due to garbage collection, and vulnerabilities such as buffer overflows. One participant also brought forward that “C programming” should be avoided; this, taken literally, would be a rather difficult proposition, given that C is still one of the most popular languages in particular in the aforementioned application domains. However, it was agreed (and that was the point of that proposal) that some of the programming constructs offered by C and similar languages are to be used with care.

The proper usage of languages like C led to the next topic, namely what we try to aim for. An MoC/language should be intuitive and familiar. However, in particular for languages that were not specifically designed with the above listed goals in mind, these languages should be used in a disciplined manner, possibly using subsets such as e.g. MISRA-C proposed by the Motor Industry Software Reliability Association. For example, ruling out dynamic memory allocation could be used to prevent memory leaks. Furthermore, one might want to use C etc. merely as an intermediate language, to be synthesized from a higher-level modeling language. The semantics should also be agnostic to available resources, as far as possible; e.g., a program should behave the same (apart from performance issues) regardless of on how many cores it is executed. Another important aspect for a language is to make a clear separation between the end user, the low-level machine implementation expert, and the compiler writer. Such separation of concern may make it possible for end-users to develop efficient systems (performance and predictable) within a shorter development time frame.

As obstacles for achieving the above goals the participants identified for example processors with unpredictable timing, as was also emphasized during earlier presentations (e.g. by Reinhard Wilhelm) during the seminar. However, it was agreed upon that the whole design flow matters, not only the execution platform. For example, a programming language should have full control over reactive control flow (concurrency and preemption), instead of handing over that responsibility to the operating system or some run time system such as the Java Virtual Machine. Typically, application-level control tasks do have a certain

level of robustness; e.g., an airplane usually can tolerate if a control output arrives after 6 msec instead of 5 msec or is missing altogether for a cycle or two. Similarly, an extra cache miss or pipeline stall should not lead to a significant change in system behavior. Design and synthesis paths must be robust, not brittle, and the used MoCs/languages should provide for that.

To conclude, some questions and remaining challenges were posed. How should time be incorporated? Do we need new languages, new MoCs? Do DSLs help? Should we have more open OSs that take hints from user/compiler? How do we achieve composability, e.g. preserve functional as well as non-functional properties of individual components?

7 Panel Discussion

Sri Parameswaran (UNSW) organized a panel discussion on adaptive isolation. The panelists were Patrick Koeberl (Intel), Tulika Mitra (NUS), Jürgen Teich (Friedrich-Alexander-Universität Erlangen-Nürnberg), and Lothar Thiele (ETH Zürich). The discussion centered around motivation for the seminar, the similarities, differences between security and predictability and their interactions/impact in the context of isolation, the future of timing predictability in the absence of commercial predictable architectures, and the tools, techniques, mechanisms for isolation covered in the seminar.

8 Acknowledgements

We would like to take the opportunity to thank and acknowledge our organization team member Ingrid Verbauwhede for her great effort in contributing brilliant ideas and suggestions on the topic of the seminar as well as to the list of participants. For very unexpected reasons, she could unfortunately not participate in the seminar.

Participants

- Davide Bertozzi
Università di Ferrara, IT
- Björn B. Brandenburg
MPI-SWS – Kaiserslautern, DE
- David Broman
KTH Royal Institute of
Technology, SE
- Samarjit Chakraborty
TU München, DE
- Sudipta Chattopadhyay
Universität des Saarlandes, DE
- Jian-Jia Chen
TU Dortmund, DE
- Albert Cohen
ENS – Paris, FR
- Ruan de Clercq
KU Leuven, BE
- Heiko Falk
TU Hamburg-Harburg, DE
- Felix Freiling
Univ. Erlangen-Nürnberg, DE
- Johannes Götzfried
Univ. Erlangen-Nürnberg, DE
- Gernot Heiser
UNSW – Sydney, AU
- Andreas Herkersdorf
TU München, DE
- Karine Heydemann
UPMC – Paris, FR
- Patrick Koeberl
Intel – Hillsboro, US
- Pieter Maene
KU Leuven, BE
- Claire Maiza
Université Grenoble Alpes –
Sait Martin d’Hères, FR
- Peter Marwedel
TU Dortmund, DE
- Tulika Mitra
National University of
Singapore, SG
- Sibin Mohan
Univ. of Illinois – Urbana, US
- Frank Mueller
North Carolina State University –
Raleigh, US
- Sri Parameswaran
UNSW – Sydney, AU
- Jan Reineke
Universität des Saarlandes, DE
- Christine Rochange
University Toulouse, FR
- Zoran Salcic
University of Auckland, NZ
- Patrick Schaumont
Virginia Polytechnic Institute –
Blacksburg, US
- Martin Schoeberl
Technical University of Denmark
– Lyngby, DK
- Wolfgang Schröder-Preikschat
Univ. Erlangen-Nürnberg, DE
- Takeshi Sugawara
Mitsubishi – Kanagawa, JP
- Jürgen Teich
Univ. Erlangen-Nürnberg, DE
- Lothar Thiele
ETH Zürich, CH
- Theo Ungerer
Universität Augsburg, DE
- Reinhard von Hanxleden
Universität Kiel, DE
- Stefan Wildermann
Univ. Erlangen-Nürnberg, DE
- Reinhard Wilhelm
Universität des Saarlandes, DE

