SAFETY & SECURITY MONITORING CONVERGENCE AT THE DAWN OF OPEN HARDWARE

Sylvain Girbal, Jimmy Le Rhun, Daniel Gracia Pérez, David Faura
Thales Research & Technology, Palaiseau, France
{sylvain.girbal, jimmy.lerhun, daniel.gracia-perez, david.faura}@thalesgroup.com

***Abstract*—The emergence of multi-core processors into the embedded world one decade ago led to the IT/OT convergence. In the last few years, a second convergence is ongoing in the domains of safety-critical and security-critical systems. Nowadays both safety protection systems and security protection systems are relying on monitoring to ensure the expected critical software behaviour. However, all these systems incur a performance overhead to fulfill the service, that could be an issue with time-critical systems.**

The safety monitoring process that was mostly involved at design time, focusing both on the software and the hardware to ensure hard real-time behaviour and propose some mitigation to faults and errors, is now also targeting the integration and deployment phases with adaptive runtime engines to deal with the timing interference issue of multi-core architectures.

The security monitoring process that was mostly used to focus on protecting against software vulnerabilities at runtime now has to consider unreliable hardware that some cyberthreats such as Spectre and Meltdown are able to exploit.

This paper proposes a short survey of existing Heath & Usage Monitoring Systems (HUMS) and Hardware Intrusion Detection Systems (HIDS) in safety-critical and security-critical systems, and their associated monitoring features.

We then promote the benefits of communalizing these monitoring features to reduce the performance impact of HUMS and HIDS systems. In this context, open hardware architectures are a major opportunity, allowing us to analyze the hardware design without black box, to seek formal proof of critical properties, to implement mechanisms for improved predictability, and to enhance hardware-level observability.

I. INTRODUCTION

During the last decades, we observed successive convergences of computing systems. It started 20 years ago with the convergence between high-performance computing (HPC) that focused on power efficiency, and the mobile market that was seeking for more performance and functionalities, both were dealing with the challenge of controlling the balance between low power and high performance.

A decade ago, thanks to the multi-core processors, a second convergence [35, 47, 37] started between the mission critical market (such as avionics, automotive, healthcare and robotics) and the mainstream consumer electronics market, also known as the IT/OT convergence (for *Information Technology* and *Operative Technology*). This convergence was fueled by the increasing requirements of the mission-critical market for computing performance, as well as the growing need of embedding more critical functionalities in the mobile devices, that started to be connected to cars or healthcare systems.

During the past years, dealing with both safety and security has become a prime requirement for embedded cyber-physical systems [65] leading to a third convergence. However, the practice is different for safety and security.

The paper is organized as follows: Section I-A and Section I-B respectively present the safety- and the security-related practices during the system development life-cycle. Section II presents the impact of the introduction of multi-core architecture on these practices, and the common safety and security trends with regard to monitoring. Section III provides a survey of existing HUMS and HIDS systems focusing on their monitoring features. Finally, Section IV presents our view on the future of monitoring techniques promoting the communalization of monitoring techniques to reduce the performance costs.

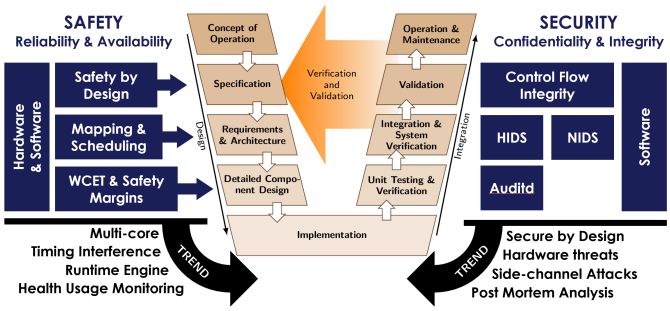

Fig. 1. Safety & Security trends with regards to monitoring

A. Safety Practices

Figure 1 present the usual V-shaped system development life-cycle, and how safety and security integrates into this scheme. Regulation standards [44, 45, 71] led the **safety critical** industry to focus mainly on the design phases to ensure reliability and availability by design [39], both at the software [72] and the hardware [73] levels.

Safety critical applications are also usually characterized by stringent real-time constraints, which are usually guaranteed by determining the application Worst Case Execution Time (WCET). This WCET computation usually relies on analysis tools based on static program analysis tools [90, 70], detailed hardware model, as well as measurement techniques through execution or simulation [38] to provide an estimated upper

bound of the execution time, introducing some safety margins as depicted in Figure 2.

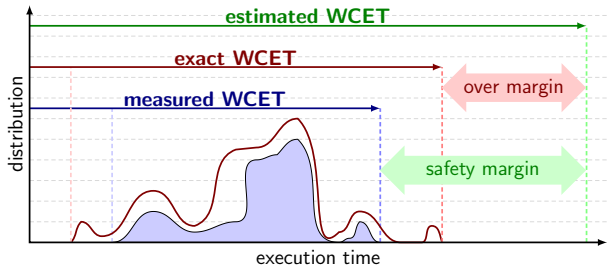

Fig. 2. Estimation of the Worst-Case Execution Time, and the over-estimation problem

The current best practices include the use of many statically defined mechanisms, such as static scheduling of periodic tasks, static memory allocation and mapping, as a way to improve the ability to demonstrate a deterministic behaviour of the system.

The introduction of multi-core architecture to safety-critical systems as presented in Section II was shown to have a significant negative impact on worst-case performance [11, 66], while enhancing average performance. The current trend to deal with this problem is to add safety nets that involve both the integration and the operational phases.

B. Security Practices

In computing systems (hardware plus software), **security** has typically been provided as specialized software executed on top of them during operation, to overcome their weaknesses, especially software vulnerabilities and intrusion detection systems [10].

For example at the device level *host intrusion detection/protection systems* (HIDS/HIPS) are used as security software applications running on the device itself, while at the network level the *network intrusion detection/protection systems* (NIDS/NIPS) are used to protect the computing systems from malicious communications.

Furthermore, external observation is nowadays not enough, requiring the integration of security solutions to be integrated at the application level or underlying layers, like operating system and even hardware. Examples are control flow integrity (CFI) [78] solutions that integrate program flow supervision to detect the unexpected exploitation of software by attackers.

A raising trend is the exploitation of hardware vulnerabilities of computing systems. Exploits such as Spectre [53] and Meltdown [58] have demonstrated that attackers can exploit vulnerabilities at hardware level. While software security solutions at different levels (compiler, operating system, firmware, etc.) have partially mitigated these vulnerabilities with some performance costs, they can only be fully and efficiently addressed at hardware level. However, the design of security solutions at hardware level is not a simple feature, and must be implemented with care to avoid introducing additional vulnerabilities [36, 56, 27].

II. Impact of Multi-Core Architecture on Safe & Secure Systems

The recent shift from single-core COTS (component off-the-shelf) to **multi-core COTS processors** for safety-critical and security-critical products was appealing both in terms of average performance and in terms of size, weight and power (SWaP) [8], actually fitting with the exponential growth in performance requirements in the embedded domain.

However, multi-core processor architectures are introducing both new sources of time variations, and new vulnerabilities: multi-core architectures are characterized by the fact that they are embedding **shared hardware resource** between the cores, as depicted in Figure 3.

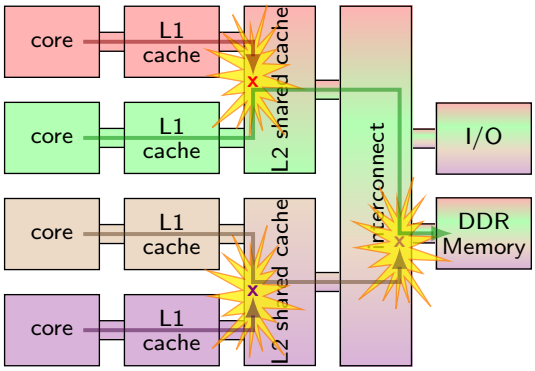

Fig. 3. Shared hardware resources & timing interference

In this figure, each core is associated with a different color, private resources (such as L1 caches) are colored with the same color as their cores, and **shared hardware resources** are represented with a shade of the involved core colors (such as L2 caches, the interconnect and the main memory).

On a multi-core COTS processor, different pieces of software will be executed on different cores at the same time. Such different software will compete electronically to use these shared hardware resources, eventually involving hardware arbiters to deal with concurrent accesses, and introducing inter-task or inter-application delay and jitter, defined as **timing interference** [31].

The lack of precise documentation coupled with the black-box aspects of these hardware arbiters impact the WCET analysis tools, that have difficulties to deal with real industrial programs running on multi-core COTS architectures [52, 64], while resource over-provisioning is no longer an option. The industry is therefore facing a trade-off between performance and predictability [90].

The literature [32] proposes several Deterministic Platform Solutions to tackle this problem, including control solutions aiming at completely preventing such timing interference and regulation solutions reducing the amount of interference below a harmful level. However, all these techniques have to circumvent the black-box aspect of the COTS architecture, at the cost of decreased performance.

On the **security** side, shared hardware resources are also a source of vulnerabilities against the confidentiality and the integrity properties, as the other cores have an opportunity to access or alter private data of co-running applications.

Multi-core processors have also impacted the security of the devices. The addition of new shared resources (e.g. bus, interconnect, L3 shared caches) or new mechanisms (or variations of existing ones) to operate in a multi-core configuration (e.g. cache coherence protocol) have introduced new attack possibilities. For example, the shared bus has been exploited to perform timing covert channels [92], the introduced cache coherence protocols have been exploited in Meltdown and Spectre variations [85] or to perform covert channel attacks [62], and dynamic frequency scaling of the different cores used to perform covert channel attacks [3].

Those examples are mainly from the IT domain, but are equally applicable to modern embedded systems with high connectivity (e.g. edge- and fog-computing), using equally complex computer architectures [88] and speculative processor cores with similar vulnerabilities (e.g. ARM v8 cores subject to Spectre attacks [7]).

Like for safety, the lack of precise documentation of those new resources and mechanisms, or their unexpected usage (as frequent in security exploits) make the defense against these attacks difficult to anticipate, endangering the confidentiality and integrity of the system and running applications.

A. Trends in Safe & Secure Systems

The introduction of multi-core architecture in safe or secure systems already led to a couple of common solutions like memory-space partitioning [71, 82], allowing to integrate several applications with different safety/security requirements in the same platform. In such a scheme, the operating system and some hardware components (MMU, TLB) are responsible for providing each software partition with its own protected memory space, actually ensuring data segregation.

Multi-cores also led to some converging trends between safety and security practice appearing in Figure 1. The timing interference problem on time-critical systems for instance has led to a set of control-based or regulation-based solutions [32] that involves the integration phase or even the operational phase with dedicated run-time engines. Health usage monitoring systems (HUMS) [63] are also targeting this operational life-cycle.

Being **secure-by-design** [60] has become an hot topic for security-critical systems with a particular focus on the design phases of the V-shaped model with approaches dedicated to security. Also, the hardware should rather not be considered as reliable anymore, but as a potential source for side-channel attacks instead [62].

A common trend for both safety and security critical systems is therefore to focus on all phases of the life-cycle from conception to operation. Also, all these new practices rely at different degrees on the ability to monitor the system.

III. Current Monitoring in Safe & Secure Systems

This section provides a survey of emerging monitoring techniques for safe & secure systems. Each subsection emphasizes a specific technique as well as the associated application domain, and how it is integrated relatively to domain-specific safety/security requirements.

A. Control Flow Integrity (CFI)

A classic security vulnerability consists in exploiting buffer overflow bugs to alter the program stack and perform code-reuse attacks [83, 42] executing malicious operations. Return Oriented Programming [80] exploits this weakness by altering the return address stored in the stack, hijacking the control flow when the function returns. Jump Oriented Programming [12, 16] later extended this threat to register corruption of direct branch targets.

Control Flow Integrity (CFI) [1, 78, 13] is a well known security approach to detects control flow hijacking, as deviation from the expected application control flow.

At compilation time, a Control Flow Graph (CFG) is generated. Each node corresponds to an uninterruptible instruction sequence (also called basic block) without any branch or return instruction. Forward edges correspond to jumps and function calls while backward edges correspond to return statements.

At runtime, to prevent control flow hijacking, we have to ensure that all jumps and all returns correspond to a legitimate edge of the CFG. However, it usually involves some code instrumentation and therefore both code intrusiveness and performance cost penalties.

1) System call instrumentation: In [49], Kadar et al. propose the less costly alternative of instrumenting/monitoring the system calls rather than the branches and returns, unexpected system call succession being also a sign of malicious code execution. They proposed a methodology to evaluate the system call instrumentation of the PikeOS real-time hypervisor in terms of performance overhead.

If they were able to keep the per system-call tracing overhead quite low with a maximum overhead of 700ns each, the high variability in number of system calls between applications as well as the critical impact of the number of context switches both made the global overhead quite complex to determine, and to be very application dependent.

Furthermore, detecting unexpected syscall succession would require to previously learn this expected succession. Applying machine learning techniques to this looks promising but will lack the certificability of being able to build an exact CFG at compile-time as required by CFI.

2) Hardware-assisted CFI: Another alternative to reduce the CFI performance overhead is to rely on the hardware rather than the software to perform branch trace collection. In [54], Kuzhiyelil and al. exploit the CoreSight hardware core coupled with hypervisor partitioning to transparently perform control flow tracing.

The CoreSight is a hardware component in ARM-v8 systems performing real-time tracing and debugging of applications. It embeds private trace FIFO queues and dedicated data paths ensuring an interference-less gathering of debugging information. Among the collectible information, the CoreSight is able to collect taken branches, thus actively monitoring the application control flow.

As depicted in Figure 4(a), a first step at compile time consists in generating the control flow graph in addition to a non-instrumented binary. During runtime, as shown in 4(b),

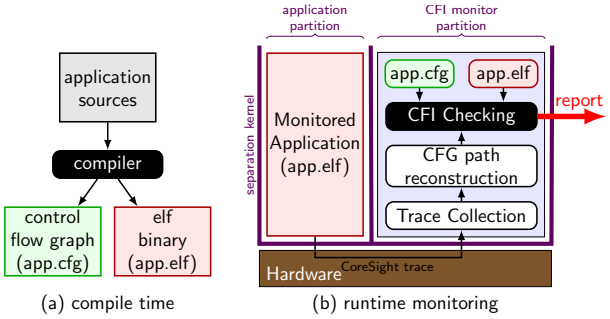

Fig. 4. Transparent control flow integrity as defined in [54]

the trace of taken branches and returns is collected through the CoreSight debug core to reconstruct the path followed in the CFG and check and report for unexpected jumps to malicious code sections.

This notion is then extended in [48] with a particular focus on mixed-critical and safety-critical systems running both critical and non-critical software. The later usually offers a larger attack surface and fewer guarantees. A particular focus in these systems is to ensure freedom from interference as required by the safety standards standards [45, 44, 72] to prevent an attack on the non-critical software from impacting the critical software.

From the security point of view, such methods show excellent detection results both in terms of false positive and false negative rates. However the performance overhead of CFI can be significant, even with a partitioned RTOS. In [54], the authors measured a worst case overhead of up to 55% for branch-heavy applications. In [48], the compromize between overhead and coverage of the detection system can be configured by the monitoring partition. Setting an overhead of 10% was shown to be sufficient to correctly classify 99% of the samples.

B. HIDS for Avionics

Safety has for long being a prime concern in avionics, however next generation aircraft systems aim at providing more and more connectivity and services to the passengers. From the security point of view, however, it continuously increases the attack surface [69]. In such a context is critical to protect aircraft software from malicious modifications of the onboard applications, and airworthiness regulations have evolved to that extent [74, 75].

The industrial process associated with avionic applications currently involve several segregated actors. The software components are developed by different *solution providers* working under computing resource budget constraints from the *platform provider*. Later these solutions are put together to build an avionic system by the *integrator*. Ideally, if the budget constraints are respected the integration should be seamless.

Protecting the aircraft software therefore involves protecting third party blackbox software components where the sources may not be available, with no possibility to add source code instrumentation or analysis as required for CFI.

Host Intrusion Detection Systems (HIDS) and anomaly detection [19, 41] have been introduced in a IT context as a way to detect such malicious modification threats, and they are now widely used for the security of information systems as a mechanism to detect abnormal or suspicious activities.

Introducing HIDS to existing safety-critical avionics products requires to take additional domain-based constraints, such as preserving the real-time constraints and the freedom of interference between components from different solution providers; keeping the footprint small, both in terms of memory footprint and resource requirement; provide explainable and reproducible results; being efficient even on blackbox components.

In [22], Damien and al. especially study how to bring HIDS to Integrated Modular Avionics (IMA) systems, considering above-mentioned specific avionics requirements. The author proposes to observe the ARINC 653 API calls performed as a model to the normal or altered behaviour of the application. This could be performed from the specific avionics RTOS by capturing both the call sequence and each call duration, but it requires a significant amount of resources to so, especially memory resources for the call trace.

Different strategies are studied to reduce the amount of data being logged, including only logging memory communication related ARINC 653 calls, or keeping only call frequency information rather than a full trace information. Several detection algorithms are also considered.

The results demonstrates that the solution keeping the whole trace, and therefore with the larger memory footprint is not the one providing the best detection results. Keeping on the more meaningful communication-related data helps with obtaining a more efficient classifier. Frequency information has also shown to be an efficient way to detect malicious modifications while requiring a much smaller memory footprint.

This approach is extended in [21] with the ability to provide a first onboard diagnosis of the anomalous behaviour, paving the way to future reactive systems with the capacity to block an attack. The author proposes the adjunction of an evolutive knowledge database as depicted in Figure 5.

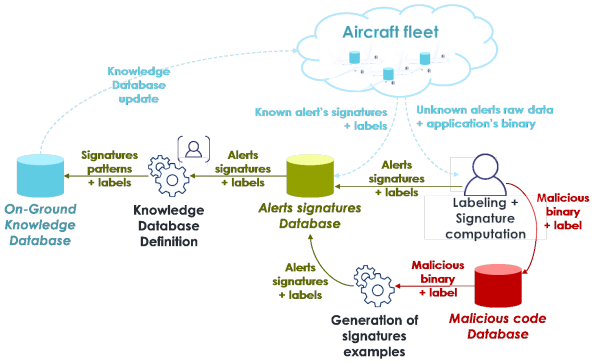

Fig. 5. Knowledge database of anomalous behaviour, as defined in [21]

This onboard knowledge database includes already known anomalous or malicious behaviour including cyber-attacks or safety-related failures, and relies on alert signatures. This database has to be regularly updated with new signatures during ground maintenance operations.

Within such a scheme, an anomaly is defined as an unknown sequence of ARINC 653 API calls, or an abnormal duration of such a call. When too many anomalies are observed during the same time frame, an alert is raised.

The anomaly signature is then searched for a match within the onboard knowledge database in order to identify the current alert and to provide a feedback message usable immediately or during later onground investigations.

The overall approach shows interesting results: In terms of error detection, it exhibits a detection accuracy and correct anomaly labeling of 87% after 70 samples. In terms of resource usage, the trace monitoring caused a +2.7% on API call runtimes and a +3.3% impact when including the early onboard diagnosis.

C. NIDS for Safety-Critical Networks

Network Intrusion Detection Systems (NIDS) are widely used in IT computing to analyze network communication traffic [10], usually performing network packet inspection and comparing them to a database of attack patterns. Such NIDS could also be combined with neural networks to increase the detection rate [30, 87].

Safety critical-systems usually embeds specific deterministic networks such as the AFDX or the CAN bus [5, 6]. In such a safety critical-context, the key properties is to guarantee the deterministic behaviour of the network and maintain a high level of integrity [84]. These mechanisms ensure a safe end-to-end transfer of information between different subsystems, preventing the propagation of network packet errors at runtime.

A664P7 [5] implements a network monitoring protocol, which tracks relevant events and communication protocol errors at the switch and end-system level. The monitoring is based on Hardware PMC dedicated to Network observability, defined in a Management Information Base (MiB) [43].

The Network Management system [2] is in charge of realizing the correlation of multi-protocol information collected from all the components to detect/localize network failure. A path of improvement is to distribute the advanced monitoring computing functionality between the aircraft systems and the airline maintenance center [25].

D. HW/SW characterization for Aerospace systems with performance counters

In the context of multi-core processors for aerospace systems, safety standards require the analysis and mitigation of undesirable contentions due to concurrent usage of shared hardware resources, called timing interference. A joint HW/SW characterization of the system behaviour is needed, with a high level of precision, including system calls and fine-grain interaction on shared resources.

To achieve such a precision level, hardware assistance is mandatory, and we can take advantage of mechanisms initially developed for performance tuning, called the performance monitoring counters (PMC). These are simple hardware registers, able to count various events within a processor core or in a SoC infrastructure, such as CPU clock cycles, the number of instructions of a certain type, cache access or misses, branch mispredicts, bus access, etc. There are typically few counters per core (e.g. 4 to 8), but up to hundreds of events to choose from.

The main advantage of this technique is the precise and timely sampling of low-level information. Time is cycle-accurate, and reading a counter only takes a couple of instructions. Activation of counters and event selection typically requires supervisor privileges, but in most cases reading the counter can be allowed in user mode, for lower overhead.

The Measurement Environment for Time-Critical Systems (METrICS) [34] is a self-contained approach to characterize multicore processor behaviour and timing interference. The basic part is a *probe*, a small piece of code inserted in an application program to perform a sampling of selected hardware events. The probe is designed to minimize timing overhead, making use of macros and inline assembly and completely avoiding system calls. Sampled counters data is stored in the main memory, with a structure designed to fit in a single cache line along with a unique probe identifier, core and process identifiers.

The probe typically takes less than 190ns to run on 1.8GHz PowerPC. The timing overhead is therefore very low, at the cost of a small source code intrusivity.

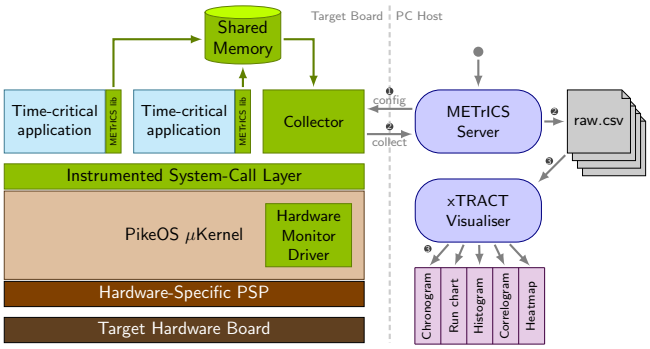

Fig. 6. METrICS architecture, as defined in [34]

The extraction of stored data is performed out of the real-time section by another component, named Collector. This is an independent partition, that is not scheduled during the measurement phase. The collector is also in charge of initialization, and data link with the host (typically using Ethernet). On-target computation is kept to a minimum, and all data is preserved for post-processing.

Several statistical techniques can then be used offline to analyze the collected data. For duration of tasks or system calls, two probes can be paired and the raw values of counters can be subtracted. Time series can be derived, as well as full histograms exhibiting the observed-WCET. Correlation between counters helps identifying possible cause of time interference.

E. Online monitoring with PMCs

As the timing overhead to collect the PMCs presented in the previous section is very low, the same principles can be extended as an online monitoring technique, as part of the health and safety usage monitoring subsystem. Multiple online interference mitigation techniques have been exploiting the

PMCs' monitoring. MemGuard [93] exploits memory accesses PMCs to monitor the number of accesses done by the different partitions in a multi-core system and perform on-board decisions guided by budget limits associated to these partitions to minimize the interference impact. Similarly, in [20] the time and instructions counters are used in addition to the memory access counters to enhance the interference control. BB-RTE (Budget-Based RunTime Engine) [33] proposes a variation of the previous two where all the shared resources PMCs, i.e. not only memory accesses, are considered to perform online interference mitigation.

To achieve the same goals without source code modification, another approach presented by Airbus in [29] makes use of an external Safety-Net processor to monitor the operational multi-core processor. As depicted in Figure 7, an external FPGA device contains a soft-core processor, connected with a high-speed link to the debug infrastructure of the multi-core. It is able to periodically access the performance counters through this link, and record the application behaviour without intrusivity.

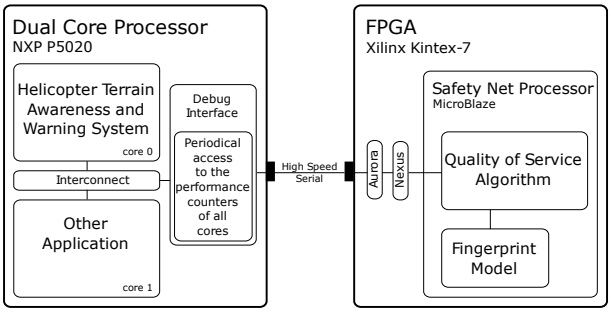

Fig. 7. Safety-Net architecture, as defined in [29]

In this case, the rate of executed instructions (in instr/μs) is the value of interest, and its evolution over the duration of a major scheduling frame. Such fingerprints can be concatenated over several time slices for a given partition, and compared between isolated and concurrent executions. The presence of timing interference can be detected by a shift of the fingerprint, as the rate of executed instruction is slightly lower and its variation pattern is slightly delayed. A slowdown of 1.5% can be reliably detected in less than 15ms.

Likely, solutions to detect security incidents exploit the PMCs monitoring techniques [28]. For example, Li [55] proposes a PMC monitoring solution able to detect a Spectre attack using the cache miss and branch missprediction PMCs available in most processors. Similarly, Chiappetta [17] exploits the L3 accesses PMC to detect side-channel attacks exploiting the caches.

These and other studies have shown the capability of PMCs usage to develop safety and security characterization and monitoring solutions. However, multiple studies [81, 86, 9] have proven that the exploitation of these same PMCs for the development of security attacks.

F. Miscellaneous Monitoring techniques

Some monitoring-base detection solutions introduce creative techniques either for trace collection or for classification.

In automotive, services used to be integrated as distinct electronic control units (ECUs) each with a specific hardware. With the multiplication of the number of services, as well as optimizing Size, Weight and Power (SWaP), ECUs are now integrated on the same hardware as virtual machines.

The HIDS introduced in the [50] position paper models the system interactions both in terms of OS service usage as well as hardware activity collected through Performance Monitor Counters. Traces are collected as words and sentences, applying Natural Language Processing techniques to predict further traces from the already collected sequence. Threat detection is then performed by comparing the actual trace to the predicted one, not requiring any form of previous offline learning.

HUMS systems on their side can consider input data beyond the software or the hardware behaviour. In [59], Airbus captures the impact of vibration in an helicopter on the mechanical components wear-out to guide out maintenance tasks.

Such an activity used to being performed with on-ground calculators during stress-test procedures. The paper proposed to shift this activity onboard.

The main asset is to implement the fundamental ability to analyze on the fly the data collected from various embedded aircraft sensors and quickly identify safety/security-related events to take the most appropriate actions. However, the authors pointed some critical missing features to collect information from the physical layer, such as a lightweight tracing and timestamping mechanism, as well as strong cycle-accurate synchronized time requirements.

The goal is going toward experimentation in onboard machine learning [46] to pave the way to predictive maintenance as part of Flight Data Monitoring [25].

G. AI usage in Monitoring

In the survey presented this section, several monitoring techniques embed different flavours of artificial intelligence: Some [34, 20, 33] are just relying on statistical analysis to compute a threshold used for outlier/anomaly detection. Some others [29, 48] rely on machine learning techniques to compute such a threshold, but are keeping the inference as a simple comparison to this threshold. Machine learning is used to a greater extent by [55, 17, 22, 21] with a more complex online inference system based on neural networks. Alternative AI approaches such as genetic algorithms or sequence prediction are also used in [22, 50]. Finally a few papers [49, 59] are considering machine learning techniques as a possible future work.

It corresponds to a recent trend of using artificial intelligence for anomaly detection, that was first introduced for IT systems [15, 57], and is now considered for embedded safety and security critical systems [51].

IV. The Future of Monitoring in Safe & Secure Systems

As illustrated by Section III, software and hardware monitoring have become prime requirements for both safety and

security, however this extra monitoring activity comes with complexity, security and performance costs. As a consequence, the convergence between safety-critical and security-critical systems would benefit from communalizing the monitoring features.

A. Multi-level aspects

Figure 8 shows the different layers composing the technology stack. This includes the user-mode applications, the domain-specific middlewares, the operating system layer, the embedded hardware SoC and the physical communication layer. The arrows show regular component interactions, for instance the operating system scheduler sets which application should be running, setting up the proper MMU entries for logical to physical address translation and flushes the hardware TLB caching the MMU.

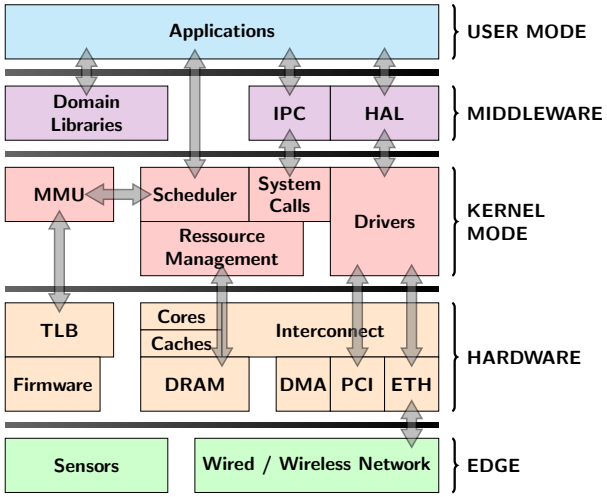

Fig. 8. Multi-layer software/hardware stack

Among all the monitoring techniques presented in the survey of Section III, many techniques are already multi-level implying several layers.

For instance, hardware-assisted CFI [54] monitors the application path behaviour from the debug module of the hardware layer. To perform a transparent, uninstrumented monitoring of the application the [21] HIDS gathers ARINC 653 call information from the operating system/middleware layers. Network monitoring has the opportunity to be performed within the hardware abstraction layer (HAL), at the Ethernet driver level, or further down on the physical link.

In fact, each layer has only limited information and lacks the semantics of the other layers: the hardware layer has efficient and immediate pipeline-related information from the Performance Monitor Counters (PMCs) such as branches or even cache misses but does not know which application, task or thread is running. This information is only available either directly from the application layer or from the scheduler of the operating system layer.

Accessing the PMCs registers is also doable from the application layer, but at the cost of additional code to be executed to capture this information and also the cost of traversal time from the application layer down to the hardware (e.g. syscalls), providing slightly outdated and noisy information.

Gathering information from multiple layers is therefore necessary to perform efficient monitoring. Pushing this concept further, we might also benefit from gathering information from multiple sources, communalizing the monitoring information from different subsystems. It will reduce the overall performance costs, as many HIDS/HUMS are accessing the same information, and provide opportunity to identify new correlations for detection. However, it comes with a set of associated challenges.

B. The Requirements for Communalizing Monitoring Information

The survey from Section III identified several challenges or specific requirements for the different detection techniques:

1) Trace collection: All previously detailed techniques optimize trace collection either in terms of intrusiveness or performance:

At user **application level**, [54, 21, 29, 55] minimize application intrusiveness, avoiding any code modification by either performing the trace collection automatically, at hardware level or externally at the cost of some performance. [34, 50] focus on minimizing time intrusiveness and optimizing performance by requiring some user instrumentation.

Some techniques perform their collection at **operating system level**, such as [54, 21, 50, 20, 93], requiring the source code of the RTOS. Instead, [34, 55] rely on a kernel driver to perform the actions requiring privileged mode.

Many collection techniques rely on **hardware level** information: [54, 29] use a specific but COTS hardware component to gather debug traces. [34, 29, 55, 50, 20, 93] are gathering the hardware-level Performance Monitor Counters.

2) Classification/detection: is usually performed outside of the monitored application, as an distinct adhoc process. To limit the impact of such a process on the monitored application, [54, 21, 33, 50, 20, 93] are relying on a separation kernel or partitioning hypervisor.

From the **hardware** point of view, [29, 59] are relying on specific onboard hardware to perform the detection, whereas [34] relies on an external host to perform the statistical analysis as a post-processing action.

3) Exploiting monitoring for side channel attacks: As stated before, most classification algorithm are performed in software from trace data extracted from the hardware [54, 21, 34, 33, 50, 55, 17]. As a consequence the monitored information is available in user-mode and therefore also available for malicious purpose to implement side-channel attacks. [29, 59] are alleviating this risks by performing this on dedicated hardware.

Ideally, to reduce the attack surface and performance cost, supervision solution should be implemented at hardware level with private resources. However, customizing hardware could be costly.

C. Open Hardware in Safe & Secure Systems and the benefits of low-level monitoring

Many of the issues faced in critical embedded systems are rooted in the incomplete or imprecise knowledge of the processor system inner operation. The high complexity of current processors necessitates abstraction for efficient usage, and many implementation details are hidden to the developer, either with each software layer or even at hardware register level. While those hidden details are specifically designed not to impact functional correctness or average-case performance, they can have a significant impact on worst-case performance that matters for critical systems.

In reaction to the increasing complexity of hardware implementation of processor systems, we observe a current rise in popularity for **Open Source Hardware**. Several institutions team up to develop more generic hardware platforms, with a common set of requirements, and share under a more or less permissive license the burden of development, verification and documentation on an accessible code base. From this point, extensions and product differentiation is possible. The open-source nature also allows security audit of the source code, and precise documentation of low-level mechanisms such as shared resource arbitration.

1) RISC-V ecosystem: One of those Open-Source Hardware initiatives, currently gathering a large momentum in the industry, is **RISC-V** [89]. As a fifth-generation processor instruction set specification, it aims at becoming the "Linux of the processors" with applications from simple IoT devices up to supercomputers. The specification is maintained by RISC-V International [76], a non-profit organization structured in several working groups and strong of more than 280 members, including all major actors of the computing industry.

The RISC-V ISA is modular and specifies 32-, 64- and 128-bits versions, with various optional extensions such as bitwise operations or hardware virtualization. In addition, industry associations such as **OpenHardware Group** [67] or **Chips Alliance** [18] focus on open-source implementations of RISC-V processors.

2) Safety & Security in the context of Open Hardware systems: Safety- and security-critical systems are niche markets, compared to mainstream computing products, and face difficulty to have their stringent requirements satisfied in the COTS market. The Open Source ecosystem allows several stakeholders to team for the specification, the implementation and the validation of processors with suitable features, whereas they would not have had the resources to do so or to influence COTS vendors individually. Within RISC-V International, these topics are notably addressed in the Security Standing Committee and the Functional Safety Special Interest Group.

Among the specific mechanisms required for critical systems, a guarantee of time-, memory-space and computer-space isolation is a common need to ensure that a dysfunctional application cannot impact other critical applications. Health monitoring, integrity checks and run-time monitoring are needed to ensure proper operation and react to errors/attacks.

In the context of multi-core processors, achieving those properties present extra difficulty as explained in Section II. However the openness of Open Hardware allows to lift the curtain on previously black-box subsystems such as the arbiters on the interconnect and other shared hardware resources, and take into account their scheduling policy in interference mitigation strategies.

Furthermore, the access to such low-level design elements offers the opportunity to perform formal validation of safety- and/or security-related properties, for example using tools such as Yosys [91].

Lastly, it provides the opportunity to add or modify specific features in the implementation, and tune hardware mechanisms to increase predictability or immunity to attacks, e.g. by reducing the sources of speculation.

3) Benefits of Open Hardware on Monitoring: In addition to the well-known advantages of openness listed above, Open-Source Hardware presents several interesting opportunities for monitoring activities, as it enables addition or modification of hardware mechanisms specifically optimized.

Low-level instrumentation can generate large amounts of data, as it operates at very fine grain. It is therefore beneficial to implement filtering capabilities in order to focus on relevant information for a given observation goal. Filter examples include address range of memory transaction, initiator core, or type of request, but very complex detection can be crafted with a combination of several monitors and filters.

Another opportunity is to propagate semantic information across system layers, that would otherwise be lost or ignored. For example, an application parameter or loop iteration could be logged along with hardware performance counters. As described in [23] this information and the filtering capabilities are actually required to perform an efficient monitoring, as those required in [55, 17].

Many low-level monitoring mechanisms focus on each processor core, but the most worrisome aspect of critical multi-core lies in the interactions at SoC-level, notably within the interconnect. Additional monitors and counters in the SoC infrastructure allow a better understanding of interference channels, and in some cases enable mitigation techniques. This is particularly interesting when combined with filtering capabilities.

A simple interrupt mechanism triggered when a performance counter reaches a given threshold allows the implementation of budget-based interference mitigation, e.g. de-schedule a task when it has exhausted its budget of memory access to ensure it is not slowing down other tasks, either because of a bug or an attack.

A recent development in this direction is the Safe Statistics Unit from BSC [14], consisting of three novel kinds of performance counters tailored for the monitoring of timing interference on multi-core processors: the Maximum-Contention Control Unit (MCCU), the Request Duration Counter (RDC), and the Cycle Contention Stack (CCS). By analyzing the traffic on a shared processor bus, it allow new ways of understanding and controlling the interactions between cores. The MCCU monitors access to a shared resource, and enforce per-core quotas. The RDC can log maximum duration of a bus transaction, and act as a watchdog. The CCS helps in the identification of the initiator at the origin of the interference suffered by each core.

Finally, a dedicated hardware mechanism can be an improvement for security concerns, as it can be designed as a separate entity, not accessible from vulnerable application software or isolated from side-channel attacks. Similarly, designs with private memories and communication channels dedicated to monitoring avoid any overhead on the system operation, and any skew in the observed behaviour.

D. Opportunities for AI-based HIDS/HUMS

Some industry domains are reluctant to introduce artificial intelligence in safety critical systems. For decades, autonomous piloting, one of the most critical function in avionics has been relying on deterministic algorithms, and the benefits in shifting to AI-based systems is not likely to cover the additional certification costs. As the monitoring subsystem runs at a lower criticality level, such a system is a good candidate to introduce AI, with lower certification costs. However embedded safety-critical systems have additional constraints to be taken into account by the AI-based systems.

In avionics, post-mortem analysis is critical to maintain flight authorization from the authorities. As a consequence it is not only a matter of successfully performing the classification/detection with an acceptable amount of false-positive and false-negative, but also a matter of identifying the root causes of a detected event. Also post-mortem analysis involves a large degree of replaying the conditions that led to the studied event. As a consequence, we expect determinism in terms of the same input causing the same answer from the AI system. Such a behaviour seems to be incompatible with continual/continuous learning [68], but more in cope with explainable AI [24] such as symbolic AI [40, 61].

Embedded systems also have stringent requirements in terms of resource usage, including processing power, real-time behaviour and memory footprint [26]. Machine learning [4, 79] seems to be in adequation with such requirements: the learning phase that builds neural networks requires both processing power and memory, but could be performed offline on external systems, while only the inference part is performed onboard with a small neural network memory footprint, and involves a constant number of multiply-add operations leading to deterministic time processing.

Another challenge for the application of AI techniques in critical systems is the low occurence of some safety/security hazards in these systems. This difficults the usage of AI techniques, particularly during the learning phase. Frugal learning [77] especially focuses on being less dependent on large collections of input data, learning from few samples with additional semantic information.

Beyond these additional challenges for embedding AI in safety/security critical systems, it also comes with new benefits and potential new markets, especially predictive maintenance, and the ability to propose more complex embedded HUMS/HIDS systems.

V. Conclusion & Future Works

In this paper, we presented a survey of emerging monitoring technologies implemented in Health Usage Monitoring Systems or Host Intrusion Detection Systems in the domain of safety-critical and security-critical devices.

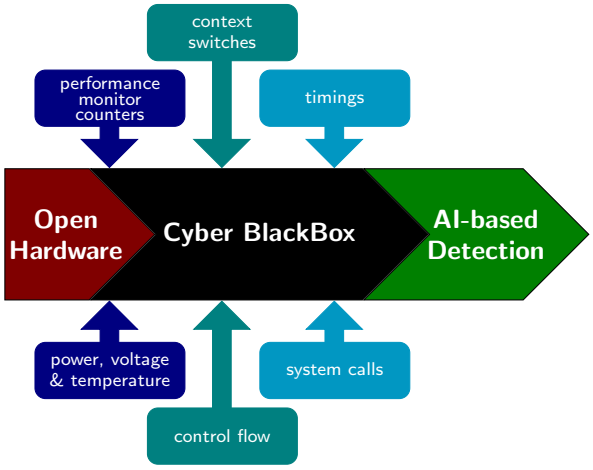

Fig. 9. Cyber BlackBox: a multi-level/muti-source approach to monitoring

We then promoted the communalization on monitoring resources in a multi-layer and multi-source approach, as depicted in Figure 9.

We foresee Open Hardware as being an enabler to implement the supervision system with dedicated software and especially hardware resources, allowing us both to capture the necessary hardware-related information, but also as a way to protect from side-channel attacks exploiting the monitoring features.

Finally, using Artificial Intelligence-based techniques looks like a promising opportunity to merge monitoring data of different natures, layers and sources to identify new correlations allowing us to detect either new safety failures or security threats, or to detect already known ones sooner.

Machine Learning in particular is usually decomposed into two phases: a preliminary learning phase, and an inference phase performing the classification. The resource-consuming learning phase could be performed offline, only leaving the inference phase to be embedded onboard and therefore reducing the resource footprint of such systems, while keeping open the explainability challenge.

The Cyber BlackBox approach presented in Figure 9 considers each monitoring source/technique as an optional plugin to the overall supervision infrastructure, so that it could be adapted the the specifics of each safety/security critical domain.

Acknowledgements

This research work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreements No 871385 and 869945.

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 2009.

[2] AIRBUS. US8190727B2, airbus patent, network management system for an aircraft, 2012.

[3] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani. DFS covert channels on multi-core platforms. In *IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017.

[4] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[5] ARINC. Aircraft data network part 7 avionics full duplex switched ethernet (afdx) network, 2005.

[6] ARINC. General standardization of can (controller area network) bus protocol for airborne use, 2007.

[7] ARM. Whitepaper: Cache speculation side-channels, version 2.5, 2020.

[8] T. G. Baker. Lessons learned integrating COTS into systems. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS '02, 2002.

[9] S. Bhattacharya and D. Mukhopadhyay. Utilizing Performance Counters for Compromising Public Key Ciphers. *ACM Trans. Priv. Secur.*, 21(1), jan 2018.

[10] E. Biermann, E. Cloete, and L. M. Venter. A comparison of intrusion detection systems. *Journal on Computers & Security*, 2001.

[11] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset, and A. Merigot. Studying co-running avionic real-time applications on multi-core COTS architectures. *Embedded Real Time Software and Systems conference*, 2014.

[12] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang. Jump-oriented programming: A new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, 2011.

[13] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer. Control-flow integrity: Precision, security, and performance. *ACM Comput. Surv.*, Apr 2017.

[14] G. Cabo, F. Bas, R. Lorenzo, D. Trilla, S. Alcaide, M. Moretó, C. Hernández, and J. Abella. Safesu: an extended statistics unit for multicore timing interference. In *2021 IEEE European Test Symposium (ETS)*, pages 1–4, 2021.

[15] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009.

[16] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.

[17] M. Chiappetta, E. Savas, and C. Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 2016.

[18] CHIPS (Common Hardware for Interfaces, Processors and Systems) Alliance. https://chipsalliance.org/. [Online].

[19] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference*, ESEC-FSE '07, 2007.

[20] A. Crespo, P. Balbastre, J. Simó, J. Coronel, D. Gracia Pérez, and P. Bonnot. Hypervisor-based multicore feedback control of mixed-criticality systems. *IEEE Access*, 2018.

[21] A. Damien, P.-F. Gimenez, N. Feyt, V. Nicomette, M. Kaâniche, and E. Alata. On-board diagnosis: A first step from detection to prevention of intrusions on avionics applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020.

[22] A. Damien, M. Marcourt, V. Nicomette, E. Alata, and M. Kaâniche. Implementation of a host-based intrusion detection system for avionic applications. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2019.

[23] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019.

[24] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018.

[25] European Union Aviation Safety Agency. Flight data monitoring on atr aircraft, 2016.

[26] M. Evchenko. Frugal Learning: Applying Machine Learning with Minimal Resources. 2016.

[27] S. Fei, Z. Yan, W. Ding, and H. Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Survey*, July 2021.

[28] J. C. Foreman. A survey of cyber security countermeasures using hardware performance counters. *CoRR*, abs/1807.10868, 2018.

[29] J. Freitag and S. Uhrig. Quality of Service for Integrated Modular Avionics (IMA) on Multicore Processors using a Safety Net Architecture. In *ERTS 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), Toulouse, France, Jan. 2018.

[30] F. Garzia, M. Lombardi, and S. Ramalingam. Artificial neural networks framework for security/safety systems management and support. In *International Carnahan Conference on Security Technology*, 2017.

[31] S. Girbal, D. Gracia Pérez, J. Le Rhun, M. Faugère, C. Pagetti, and G. Durrieu. A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In *Proceedings of the 34th Digital Avionics Systems Conference*, DASC'2015, 2015.

[32] S. Girbal, X. Jean, J. Le Rhun, D. Gracia Pérez, and M. Gatti. Deterministic Platform Software for hard real-time systems using multi-core COTS. In *Proceedings of the 34th Digital Avionics Systems Conference (DASC)*, 2015.

[33] S. Girbal and J. Le Rhun. BB-RTE: a Budget-Based RunTime Engine for Mixed & Time Critical Systems. In *Embedded Real Time Software and Systems*, ERTS '18, 2018.

[34] S. Girbal, J. Le Rhun, and H. Saoud. METrICS: a measurement environment for multi-core time critical systems. In *Embedded Real Time Software and Systems*, ERTS '18, 2018.

[35] S. Girbal, M. Moretó, A. Grasset, J. Abella, E. Quiñones, F. J. Cazorla, and S. Yehia. On the convergence of mainstream and mission-critical markets. In *50th IEEE Design Automation Conference (DAC)*, 2013.

[36] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida. ASLR on the line: Practical cache attacks on the MMU. In *NDSS*, 2017.

[37] B. Gyselinckx, R. Vullers, C. Van Hoof, J. Ryckaert, R. F. Yazicioglu, P. Fiorini, and V. Leonov. Human++: Emerging technology for body area networks. In *2006 International Conference on Very Large Scale Integration*, 2006.

[38] R. Heckmann and C. Ferdinand. Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation. In *Proceedings of the conference on Design, Automation and Test in Europe*, DATE'05, 2005.

[39] C. Hobbs. *Embedded software development for safety-critical systems*. CRC Press, 2019.

[40] V. Honavar and L. Uhr. Symbolic artificial intelligence, connectionist networks & beyond. 1994.

[41] N. Hubballi, S. Biswas, and S. Nandi. Sequencegram: n-gram modeling of system calls for program based anomaly detection. In *3rd International Conference on Communication Systems and Networks*, 2011.

[42] A. Humayed, J. Lin, F. Li, and B. Luo. Cyber-physical systems security—a survey. *IEEE Internet of Things Journal*, 2017.

[43] IETF. Rfc4293: Management information base for the internet protocol (ip), 2006.

[44] International Electrotechnical Commission. IEC 61508: Functional safety of electrical, electronic, or programmable electronic safety-related systems, 2011.

[45] International Organization for Standardization (ISO). ISO 26262: Road Vehicles – Functional Safety, 2011.

[46] ITU-T Focus Group on Aviation Applications of Cloud Computing for Flight Data Monitoring. Existing and emerging technologies of cloud computing and data analytics. 2016.

[47] A. Jahn, M. Holzbock, J. Muller, R. Kebel, M. de Sanctis, A. Rogoyski, E. Trachtman, O. Franzrahe, M. Werner, and F. Hu. Evolution of aeronautical communications for personal and multimedia services. *IEEE Communications Magazine*, 2003.

[48] M. Kadar, G. Fohler, D. Kuzhiyelil, and P. Gorski. Safety-aware integration of hardware-assisted program tracing in mixed-criticality systems for security monitoring. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[49] M. Kadar, S. Tverdyshev, and G. Fohler. System calls instrumentation for intrusion detection in embedded mixed-criticality systems. In *CERTS*, 2019.

[50] M. Kadar, S. Tverdyshev, and G. Fohler. Towards host intrusion detection for embedded industrial systems. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020.

[51] G. Kasparaviciute, M. Thelin, P. Nordin, P. Söderstam, C. Magnusson, and M. Almljung. Online encoder-decoder anomaly detection using encoder-decoder architecture with novel self-configuring neural networks & pure linear genetic programming for embedded systems. In *Proceedings of the 11th International Joint Conference on Computational Intelligence*, IJCCI 2019, page 163–171, Setubal, PRT, 2019. SCITEPRESS - Science and Technology Publications, Lda.

[52] R. Kirner and P. Puschner. Obstacles in worst-case execution time analysis. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008.

[53] P. Kocher et al. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[54] D. Kuzhiyelil, P. Zieris, M. Kadar, S. Tverdyshev, and G. Fohler. Towards transparent control-flow integrity in safety-critical systems. In *ISC*, 2020.

[55] C. Li and J.-L. Gaudiot. Online Detection of Spectre Attacks Using Microarchitectural Traces from Performance Counters. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018.

[56] M. Li, Y. Zhang, Z. Lin, and Y. Solihin. Exploiting unprotected I/O operations in AMD's secure encrypted virtualization. In *Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.

[57] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich. A survey on anomaly detection for technical systems using lstm networks. *Computers in Industry*, 131:103498, 2021.

[58] M. Lipp et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium*, Baltimore, Aug. 2018.

[59] M. Lo, N. Valot, F. Maraninchi, and P. Raymond. Real-time on-Board Manycore Implementation of a Health Monitoring System: Lessons Learnt. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, Jan. 2018.

[60] S. Longari, A. Cannizzo, M. Carminati, and S. Zanero. A secure-by-design framework for automotive on-board network risk analysis. In *2019 IEEE Vehicular Networking Conference (VNC)*, 2019.

[61] J. Mattioli, P.-O. Robic, and T. Reydellet. L'intelligence artificielle au service de la maintenance prévisionnelle. 07 2018.

[62] C. Maurice, C. Neumann, O. Heen, and A. Francillon. C5: Cross-cores cache covert channel. In *DIMVA*, 2015.

[63] S. Mekid. IoT for health and usage monitoring systems: mitigating consequences in manufacturing under cbm. In *18th IEEE International Multi-Conference on Systems, Signals & Devices (SSD)*, 2021.

[64] E. Mezzetti and T. Vardanega. On the industrial fitness of WCET analysis. In *Proceedings of the 11th International Workshop on Worst Case Execution Time Analysis (WCET2011)*. 2011.

[65] H. Mun, K. Han, and D. H. Lee. Ensuring safety and security in can-based automotive embedded systems: A combination of design optimization and secure communication. *IEEE Transactions on Vehicular Technology*, 2020.

[66] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. *European Dependable Computing Conference*, 2012.

[67] OpenHW Group: Proven processor IP. https://www.openhwgroup.org/. [Online].

[68] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 113:54–71, 2019.

[69] S. Parkinson, P. Ward, K. Wilson, and J. Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[70] P. Puschner and A. Burns. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems*, 2000.

[71] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (IMA) development guidance and certification considerations.

[72] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-178B: Software considerations in airborne systems and equipment certification, 1992.

[73] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-254: Hardware considerations in airborne systems and equipment certification, 1992.

[74] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-326: Airworthiness security process specification, 2010.

[75] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-356: Airworthiness security methods and considerations, 2015.

[76] RISC-V International. https://riscv.org/. [Online].

[77] H. Sahbi, S. Deschamps, and A. Stoian. Frugal Learning for Interactive Satellite Image Change Detection. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pages 2811–2814. IEEE, 2021.

[78] S. Sayeed, H. Marco-Gisbert, I. Ripoll, and M. Birch. Control-flow integrity: Attacks and protections. *Applied Sciences*, 2019.

[79] J. Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[80] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[81] M. Spisak. Hardware-Assisted Rootkits: Abusing Performance Counters on the ARM and X86 Architectures. In *Proceedings of the 10th USENIX Conference on Offensive Technologies*, 2016.

[82] R. Strackx, F. Piessens, and B. Preneel. Efficient isolation of trusted subsystems in embedded systems. In *International Conference on Security and Privacy in Communication Systems*, 2010.

[83] L. Szekeres, M. Payer, T. Wei, and D. Song. SoK: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, 2013.

[84] P. Toillon, P. B. Champeaux, D. Faura, W. Terroy, and M. Gatti. An optimized answer toward a switchless avionics communication network. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015.

[85] C. Trippel, D. Lustig, and M. Martonosi. MeltdownPrime and SpectrePrime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols, 2018.

[86] L. Uhsadel, A. Georges, and I. Verbauwhede. Exploiting Hardware Performance Counters. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2008.

[87] D. W. F. L. Vilela, A. D. P. Lotufo, and C. R. Santos. Fuzzy artmap neural network ids evaluation applied for real ieee 802.11w data base. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[88] X. Wang, Y. Yang, and Y. Han. Enforcing security for real-time multicore embedded system. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 1551–1556. IEEE, 2018.

[89] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic. The RISC-V instruction set manual, 2014.

[90] R. Wilhelm et al. The worst case execution time problem, overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 2008.

[91] C. Wolf. Formal verification with symbiyosys and yosys-smtbmc. *URL http://www.clifford.at/papers/2017/smtbmc-sby/slides.pdf*, 2017.

[92] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, 2012.

[93] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. R. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2013.