A Runtime Security Monitoring Architecture for Embedded Hypervisors

Henry Hui*, Kieran McLaughlin, Fahad Siddiqui, Sakir Sezer, Sena Yengec Tasdemir, Balmukund Sonigara
Queen's University Belfast
Belfast, United Kingdom
{h.hui,kieran.mclaughlin,f.siddiqui,s.sezer,s.yengectasdemir,b.sonigara}@qub.ac.uk

***Abstract*— Technical advances in automation, embedded hardware and virtualization nurtured the rapid development of safety-critical embedded systems. Avionic and Automotive are two of the domains that have seen significant development in recent years. However, the consideration of security of these embedded controllers often lags behind other functional improvements. This paper presents a brief threat landscape of embedded controllers and proposes a runtime security monitoring architecture aiming to protect these systems against threats that emerge during runtime. To facilitate rapid prototyping and encourage correct adaptation of security, similar to software development toolchains that guide development, the proposed architecture aims to provide a means for security monitoring that can be applicable to generic platforms.**

***Index Terms*— Embedded Systems, Hypervisor, Runtime security monitoring, Security.**

I. INTRODUCTION

Embedded controllers are being utilised in virtually every industrial domain for their ever-improving hardware capability and connectivity. This is often accompanied by virtualisation, which provides embedded systems with greater flexibility to improve and add functionalities. One of the widely used virtualisation techniques is the use of a hypervisor. In terms of security, hypervisors can provide spatial and temporal isolation between system resources, which allows segregated execution of processes with different criticality. In general computing, runtime security along with virtualisation is a relatively well-researched topic [1]. This is in contrast to the field of safety-critical embedded systems, e.g. avionic and automotive systems, despite safety-critical systems being one of the most high-impact targets for cyber-attacks. Moreover, since the development goals of these systems are moving towards automation, with implied safety constraints, the corresponding software and hardware become increasingly complex. There is a general consensus that the more complex a system is, the more error-prone the design and implementation process, which generates a larger surface of threats. As described in Section II, there is a lack of attention in the literature towards providing runtime security that can be applied to safety-critical systems supported by hypervisor technologies. There are several challenges for security monitoring solutions in this particular domain, including: comparatively low processing power, since these systems are often designed to run on batteries; prioritization of safety-driven factors such as resilience and availability over performance; systems with different criticality have to share the processing system; legal/standard requirement compliance, like ISO26262 [2] and DO178C [3]; and existing approaches, especially those based on Trusted Execution Environments, often require the use of platform-specific features.

As part of the development of a toolchain [4], which aims to address the complexity of safety-critical systems (and beyond the scope of this paper), this work focuses on exploring solutions to tackle the emerging runtime threats against safety-critical embedded systems that operate using hypervisors. The proposed architecture also provides a way to ensure system integrity at runtime using hardware independent features. The contributions of this work are: to propose a Runtime Security Monitoring (RSM) architecture using an open-source and verifiable hypervisor; to provide integrity and confidentiality for data-at-rest and during communication; and to present an architecture that can be applied to safety-critical systems directly with segregation of different data. Although the presented architecture originally targets safety-critical embedded systems, with avionic and automotive being use-cases of this work, it also provides general runtime security monitoring techniques for wider embedded systems.

This rest of the paper is structured as follows: Section II describes the threats and attack vectors faced by safety-critical systems, existing runtime monitoring techniques, and the requirements for a hypervisor. Section III presents the architecture of the proposed RSM. Section IV compares the proposed architecture with related work. Section V summarises the paper.

II. BACKGROUND AND RELATED WORK

Safety-critical embedded systems refer to systems where failures have consequences involving the endangerment of

people, damage to property or negative impact to the environment [5]. Although these systems usually require extensive safety verification and approval before reaching the market, security is often considered as add-ons to the systems. One of the possible solutions for the design challenges is to utilise RSM. However, RSM remains in its infancy in the embedded domain. To provide a background of this research, a brief discussion of the general and use-cases specific threats and relevant RSM technologies used in the IT domain will be discussed. In addition, despite the lack of directly applicable work in previous literature, some RSM related work on the use-cases will be outlined.

A. Threat Landscape

Some of the threats targeting embedded systems are categorised by [6]. Figure 1. illustrates the attack vectors on safety critical systems. To gain entry to a system, attackers typically are equipped with any of the following capabilities. 1) Exploitation of internet facing devices: Adversaries scan the internet for exposed devices and exploit well-known or zero-day vulnerabilities. Generally, the attackers do not need access privileges for performing these attacks. 2) Local or remote access: Adversaries gain logical access to a device, either locally (e.g. through direct memory access) or remotely (e.g. from the internet). This allows immediate attacks that violate control flow integrity or to perform long time reconnaissance and data exfiltration. Often the intrusion can be performed with user privileges only. 3) Direct physical access and attacks in physical proximity: Direct physical access means an attacker gains access to the device and can potentially make physical modifications of the devices or perform physical DoS attacks. On the other hand, some attacks only require the malicious actors to be in close proximity of the devices, e.g. jamming or eavesdropping of wireless communication. 4) Others: There are other less utilised but equally impactful ways an attacker can attack a device. An example is to misconfigure legitimate software that causes the device to be vulnerable. An attacker with any one, or a combination, of the above capabilities can perform malicious actions to the embedded system [6]. This can be summarised into three categories: side-channel attacks [7], data injection attacks [6], [8] and memory-based code injection [9]–[12].

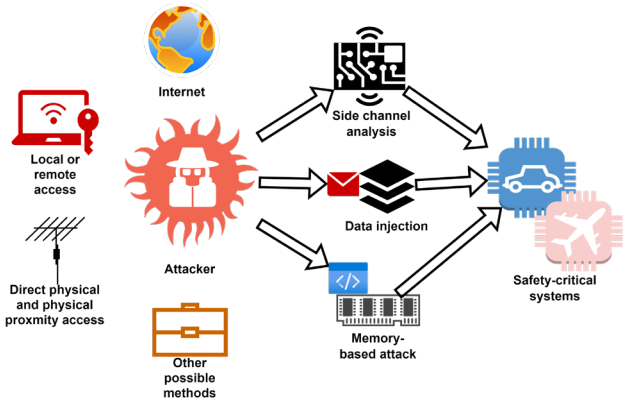

Figure 1. Attack vectors of safety-critical systems

B. Runtime Security Monitoring

In the specific domain of hypervisor technologies, [13] suggested a monitoring technique that can keep executables in check throughout runtime. Two modes of operations are proposed: user mode and kernel mode. In user mode, each application, or code segment, is signed at page level, which is then stored in a database. In contrast, kernel modules in kernel mode are linked in the running kernel upon loading, where information is used for monitoring. The kernel can also modify the loaded code. In both modes, when the memory pages for the applications are accessed by a virtual machine (VM), a vm-exit event (an event that behaves like an interrupt and hands over control to the hypervisor) will be triggered. If the event is triggered by a write access, the hypervisor will remove the execution right of the VM for the applications and allow the write action. When the trigger is caused by execution, the page storing the application will be checked for integrity and the VM for access privilege. If a violation is found, the hypervisor will inject a general-protection fault to the guest OS and the guest OS will stop the process. In other words, the general protection fault causes the system to freeze until the violation has been dealt with. However, this is clearly not appropriate for safety-critical systems. A similar mechanism, Hypersafe, is proposed by [14]. Hypersafe not only monitors the write and execute access of the hypervisor memory but is designed to ensure mutual exclusivity of either access of a memory page. By default, all memory pages are read only until a valid write request is received. After modifications, security policies are used to checked against the written area for conformity. Hypersafe also restricts pointer indexing. Through static code analysis for direct function calls and monitoring call, jmp or ret instructions, a table can be created for each function call. This information can be used to ensure control flow integrity by monitoring the invocation of functions. Another hypervisor based architecture, HyperKRP is proposed by [15]. HyperKRP utilises a small hypervisor. During system startup, HyperKRP loads the trusted boot process and requests a block of memory for the mechanism, before importing all protection policies. The mechanism also prohibits write access to the kernel memories. With a list of precompiled rules, once an vm-exit event is logged from execution of kernel objects, the hypervisor looks up the list of rules and determines whether the action is allowed. Since the action performed in kernel space and the hypervisor is strictly control, the mechanism has the ability to tolerate injection of malicious code in the user space.

Some runtime monitoring techniques are based on the concept of Trusted Execution Environments (TEEs). TEEs utilise the capability of the hardware and instruction set architecture (ISA), like the Intel Software Guard Extensions, ARM TrustZone and RISC-V Physical Memory Protection, to provide memory isolation for security. Keystone [1] presented an architecture surrounding the idea of a customisable TEE. A customisable TEE provides different capabilities based on stakeholders' input, like hardware manufacturers, programmers, and users. Based on the seL4 microkernel, Keystone provides a framework to create a system with features like separation of memories, memory page management and runtime monitoring. SeRoT [16] on the other hand provides most of what Keystone provides, in addition to

sensitive data encryption and checks the value returned by system components with different trust.

Regarding the use-cases of interest in this work, there are inherently major (often legal) safety requirements for systems, where the ongoing innovation towards automation has to take this into account. However, there is limited research regarding runtime security of such systems, which mainly focus on threat analysis, security challenges, and security evaluations of existing operating systems [17]–[20].

In the avionic domain, the MPSoC hypervisor is reviewed by [21], which takes safety, security and performance into account. It is stated that currently, safety requirement in this space, specifically ARINC 653, requires system partitioning for safety reasons. This makes hypervisor-based architectures fit in the paradigm nicely. The least privilege approach is adapted for assigning drivers to different partitions, which also means the layer with the most privilege, i.e. the hypervisor, need the strictest assurance and security. However, the work only reviews existing technologies for potential application. In the automotive domain, [22] presented an approached to secure AUTOSAR based systems using built-in functionality of the software. AUTOSAR is originally released by a development partnership of the automotive industry. The goal of AUTOSAR is to provide a common way to manage electronic and software components to allow reuse and evolutions, which at the time were limited by re-development and validation. The proposed work by [22] provides information about known problems of AUTOSAR using existing AUTOSAR tools. However, from a security perspective, this assumes the implemented AUTOSAR features are secured, and the systems being protected are not compromised at a lower level. Moreover, this approach does not address the problem caused by virtualisation.

C. *Hypervisor requirement*

As mentioned, hypervisors are extensively used in the IT domain to provide spatial and temporal separation of processes. This is ideal for safety-critical embedded systems where processes have a mixed-level of criticality. However, additional capabilities and requirements are essential when adopting hypervisors for the use-cases of this work:

1) Due to the limitations of embedded systems, e.g. low processing power, the hypervisor has to be small in size.

2) To facilitate real-time processes, the overhead of IPCs should be low, while providing security.

3) Support for mixed criticality.

4) Minimal dependence on hardware/software platform to reduce the surface of threats.

Well-known hypervisors like XEN and KVM depend on the Linux kernel, where verification for functional correctness is infeasible, if not impossible, due to the size of the codebase. The size of the kernel also increases the surface of threats. seL4 on the other hand provides a solution for this problem, while addresses the requirements above.

seL4 [23], [24] is a microkernel that supports virtualisation and can be used as a hypervisor. Multiple threads can be run on the hypervisors, where the threads can be applications, wrappers for the hardware or VMs. Detailed information for seL4 can be found in [23], however below is the justification of selecting seL4 as the basis of the proposed RSM architecture.

I) The codebase of seL4 is small, at round 10,000 lines-of-code, making it ideal for embedded applications. It is also formally verified for implementation correctness [25] and security was proven to be enforced [26]. This sets an important foundation for safety-critical embedded systems.

II) seL4 by design has a low overhead for Inter-Process Communication (IPC) since seL4 treats IPC as an invocation of other system components. [24], [27] shows that seL4 costs about 10-20% above the hardware kernel entry event, which is much better than other microkernels. seL4 also provides capability-based security on IPCs. A "capability" in seL4 is an object reference with encoded access rights. Upon an operation, e.g. a function call, the kernel will only allow access for an object reference with the correct access right.

III) The microkernel supports mixed-criticality. Different components of a system may have different consequences of failure, e.g. the brakes of a vehicle vs the infotainment system. seL4 provides a way to efficiently assign resources for different components for different real-time requirements and ensure the failure of a component does not affect other components with higher criticality [28]. An analysis of its worst-case execution time is also available and the details are provided in [29], [30].

In addition, seL4 has been used in real-world applications, for example, a helicopter run on a seL4 based system has proven the viability of the hypervisor being run on safety-critical embedded system [31]. Given the potential, this work extends the research by proposing an architecture to streamline the implementation and monitoring of runtime security.

III. Runtime Security Monitoring Architecture

The now proposed architecture leverages seL4's capability-based security and aims to provide an extra layer of security for safety-critical embedded systems. Moreover, the design of the architecture also emphasises protecting the system "when" (not "if") some of the system components are compromised. A few assumptions are made for the proposed mechanism:

- A hardware trusted platform module and secure boot functionality is available.
- Availability of a trusted service repository for critical and safety related software (typically from vendors).
- Firmware updates will not be applied during normal operations.

Fig. 2 shows the overall proposed architecture. Note that while seL4 recommends extracting and isolating functionality of a bigger VM into multiple smaller independent components, the components are generalised and are presented as *enclaves* in this work. In practise, the seL4 guidance should be followed to provide component segregation when possible.

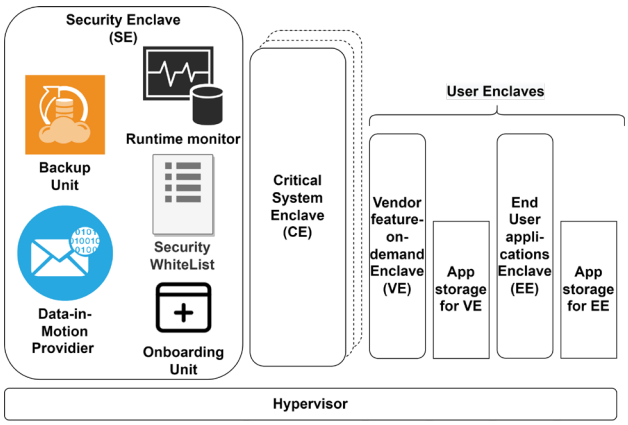

Figure 2. Runtime Security Monitoring Architecture

The proposed architecture includes two *System Enclaves* by default, the Security Enclave (SE) and the Critical systems Enclave (CE). For optional and user specific components, two optional *User Enclave*s are available: the Vendor feature-on-demand Enclave (VE) and End User applications Enclave (EE). More information is provided in due course. A specific security concept being considered in the RSM is the Data-at-rest (DAR) security. DAR means data either stored in the memory or persistent storage. This can be applications, telemetry data, user data, or cryptographic keys. This information should be encrypted in the system. Detailed information of the architecture is given below.

A. *User Enclaves*

VEs and EEs are optional and non-critical components that serve different purposes for users in the way that VEs provide functionality that interact with the hardware of the systems, for example, custom functionality or features-on-demand (e.g. enable driver assistance system). This is where users can choose non-critical system functions, provided by vendors or trusted third parties. EEs on the other hand provide limited access to the hardware and the privilege can be revoked anytime by the runtime monitor, e.g. the infotainment system of a self-driving vehicle where the screen access can be revoked on safety or security events in favour of important actions.

In addition, if a thread in the user enclaves is a VM, an associated Application Storage (AS) is required. This allows the separation of the VM image and installed application, where the invocation of an application requires a seL4 IPC pointer with the right capability, providing the runtime monitor with the ability to stop potential malicious calls and that the AS can be removed or restored at will.

B. *Critical System Enclave (CE)*

The CE encapsulates the essential component of a safety-critical embedded system, for example, the brake system of an autonomous road vehicle or the collision avoidance system of an unmanned avionic vehicle. A major restriction on the CE is the inability to perform changes to the included binaries during runtime. The justification of this decision is perhaps obvious; however, this also calls for additional validation and verification before the system is deployed to ensure functional correctness. The integrity of the CE is also examined regularly. Other behaviours of the CE are also strictly monitored by the Security Enclaves, for example, during maintenance, the installation or the updating of new software be performed by the onboarding partition in the SE, and the communications with all components are monitored by the Data-In-Motion provider.

C. *Security Enclave (SE)*

The SE contains a collection of security components provided by the proposed architecture.

1) *Data-In-Motion (DIM) Provider*

Data-In-Motion refers to the transmission of data from one endpoint to another, either internal (between software services) or external (to the internet). The DIM provider provides the cryptographic components that facilitate secure inter-component and external communications. Both data encryption and data signatures are required for each communication. The private keys for data signatures and the exchange of symmetric keys (e.g. through Diffie–Hellman key exchange) would be set up during system initialisation. The DIM provider also manages the Inter-Process Communication (IPC) and the networking interface (e.g. drivers for cellular broadband and wireless radio). It should be remembered again that seL4 IPC are designed similarly to an “object pointer” in programming terms, where the contents of the IPC are like parameters of a function. Moreover, as part of the capability-based security, IPCs are only available to authorised users. In the case where other hypervisor is preferred, capability-based security can be implemented as a custom IPC.

The DIM provider further protects the system against reconnaissance and spoofing from an infected components by directing all IPC communications to the DIM provider, where the instructions are logged before being forwarded to the intended endpoint. Therefore, unless it is intended by design, the sender of the IPC has no knowledge of other components and functionalities that exist in the system. Moreover, since the IPCs are encrypted and signed, even if a malicious component gained access to the system’s memory, the details of the IPC cannot be obtained or amended. Communications intended for external endpoints will be secured through existing technologies like TLS. These endpoints can either be the trusted service repository or the wider internet, which can only be accessed by the Onboarding Unit and the End User applications unit respectively.

2) *Onboarding Unit (OU)*

The OU is designed to handle the installation, the update and uninstallation (offboarding) of the software. This can be Features-on-Demand (FoD) or system firmware updates. As mentioned earlier, it is assumed that firmware updates will only be applied outside of normal operations, e.g. during maintenance. In addition, it can only be initiated by the CE. On the other hand, FOD on/off-boarding and updates refers to the installing, removing and updating of user software. FOD are provided by the vendors or third parties through a trusted service repository. The DIM provider is responsible for the security of the communication, while the OU logs the details of the software being installed, e.g. the hash of the binary, and checks the system component for onboarding privileges. The

same is also done by the OU when offboarding the unit to prevent unauthorised removal of software.

3) Security Whitelist

The security whitelist is another integral part of the architecture. It facilitates the initialisation of the systems and security monitoring at runtime. Each entry of the whitelist includes the digital signature of the application binary and any access privilege information. The digital signatures allow the runtime monitor to check and verify the integrity of the binary on important events, like system start-up and FOD updates. The access privilege information records which system components are allowed access to a particular binary.

4) Backup Unit (BU)

Safety-critical systems often require redundancy or a safety net. For example, [32] proposed a hardware redundancy measures for Unmanned Aerial Vehicles. The BU in this work provides a way to restore system to the last known good state. Only the Runtime Monitor (RM) has the authority to invoke the BU. While other capabilities of the RM will be described in the next section, here the RM scans each system unit for anomalies and attempts to terminate them. The BU therefore is intended to be a last resort in the case that RM loses control of a system unit (e.g. a malware infected EE gained control of the CU and cannot be terminated). This is achieved by the BU managing a copy of the CU and the base images of VU and EUU. In terms of the corresponding app storage, limited (due to resource constraint) important functionality in VU can be restored as part of a backup, while less important applications can either be re-onboarded according to the whitelist and the last known good state of the system, or simply be ignored. Given interruptions are likely to be caused by this action, system designers should thoroughly analyse and evaluate the priority of the safety and functional components, which should already be part of the normal design process. In the case where redundant hardware is available, e.g. as mentioned in [32], the system can be up and running again with minimal effect to its operation.

5) Runtime Monitor (RM)

The RM actively monitors the communications facilitated by the DIM provider, by performing regular system integrity checks, monitoring process behaviour in each system unit, and handling security incidents. As mentioned previously, every internal and external communication is delivered through the DIM provider. With capability-based security provided by seL4, the privilege of each component is checked on every IPC. Abnormal attempts will be flagged and may be blocked by the runtime monitor. In addition, the runtime monitor performs regular data-at-rest checks, monitoring user enclaves for app invocation, and has the ability to terminate processes that are flagged abnormal in the UEs.

IV. ARCHITECTURAL COMPARSIONS

The proposed architecture provides secure IPC and external communications, data-at-rest integrity and confidentiality, independence from ISA, architectural separation between safety-critical and optional user facing systems, and runtime security monitoring. Table I. compares the presented architecture with related work from Section II. There is a lack of research regarding runtime security monitoring, especially in the avionic domain, where [21] only provides a review of existing technologies that can be used for monitoring purposes. In another type of safety-critical system, [22] uses AUTOSAR, an automotive domain specific software originally design for modelling electrical and software components, to provide monitoring capabilities. As mentioned, other literatures related to the use-cases, like [17]–[20], are predominantly survey-based research where no technical work was proposed. The architecture presented in this work addresses this gap by providing a ready-to-use approach that considers the requirements of safety-critical systems. Among the architectures discussed, only the presented work, [15], and [22] are ISA independent architectures, which do not require the use specific hardware instructions. This means that the architecture can be applied in addition to any ISA provided security mechanism, instead of that being the only line of defence. In addition, all except [21] provide integrity for data-at-rest, while IPC integrity is lacking for [21] and [22]. This reflects previous experience on defending attacks like data and command injection, as mentioned in Section II.A. However, confidentiality of inter-component communication is only covered by this work and [16]. Given the importance of safety-critical systems, it is essential to provide the facilities for data encryption used by the exchange of sensitive data, such as cryptographic keys. Another benefit of IPC encryption is to provide an extra layer of defence when applying access control on calls invocation, which is provided by this work, [14], [15] and [1]. Finally, this work uses a verified kernel as the basis of the security monitor, which allows the trust obtained by the kernel from the root-of-trust (e.g. hardware TPM with secure boot) to be transferred to the security monitor correctly. On top of providing hardened security, the security architecture can potentially be verified for functional correctness.

TABLE 1. COMPARSIONS OF FEATURES BETWEEN THE PROPOSED ARCHITECTURE AND RELATED WORK.

Architectural features	[13]	[14]	[15]	[1]	[16]	[21]	[22]	This work
Hypervisor based	✓	✓	✓	✓	✓	✓	×	✓
ISA independent	×	×	✓	✓	×	×	✓	✓
Data integrity	✓	✓	✓	✓	✓	×	✓	✓
IPC Confidentiality	×	×	×	×	✓	×	×	✓
IPC Integrity	✓	✓	✓	✓	✓	×	×	✓
IPC Access Control	×	✓	✓	✓	×	×	×	✓
Based on verified kernel	×	×	×	✓	×	×	×	✓

V. CONCLUSIONS

This work proposes a novel architecture that allows runtime monitoring to be directly deployed on safety-critical embedded systems. While this work opted for the seL4 platform, the architecture can be applied on others. By adopting a multi-enclave design as proposed, components with different criticality can be protected by access segregation. In addition, the integrity of communications among components

is ensured along with tamper-proof data storage. System invocations are monitored for privileged use and the security is hardened by encryption of IPC. When compared to the literature, where some of the latest approaches required functionalities provided by ISA [1], namely TEEs, the proposed work allows otherwise. This also provides the possibility for retrofitting the architecture to systems that do not have the required capabilities, like TEEs, while providing extra hardening to newer systems if TEEs are available.

ACKNOWLEDGEMENT

This research work was funded by the European Union's Horizon 2020 Research and Innovation Programme under Grant 957210.

REFERENCES

[1] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, 'Keystone: An open framework for architecting trusted execution environments', in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[2] O. I. de Normalización, *ISO 26262: Road Vehicles : Functional Safety*. ISO, 2011.

[3] RTCA, Inc, 'DO-178C "Software Considerations in Airborne Systems and Equipment Certification"'. Jan. 05, 2012.

[4] T. Dörr *et al.*, 'A Behavior Specification and Simulation Methodology for Embedded Real-Time Software', in *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Sep. 2022, pp. 151–159.

[5] J. C. Knight, 'Safety critical systems: challenges and directions', in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, May 2002, pp. 547–550.

[6] D. Papp, Z. Ma, and L. Buttyan, 'Embedded systems security: Threats, vulnerabilities, and attack taxonomy', in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, Jul. 2015, pp. 145–152.

[7] F. Menichelli, R. Menicocci, M. Olivieri, and A. Trifiletti, 'High-Level Side-Channel Attack Modeling and Simulation for Security-Critical Systems on Chips', *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 3, pp. 164–176, Jul. 2008

[8] M. Hasan and S. Mohan, 'Protecting Actuators in Safety-Critical IoT Systems from Control Spoofing Attacks', in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things - IoT S&P'19*.

[9] 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, pp. 552–561.

[10] R. Hund, T. Holz, and F. C. Freiling, 'Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms', p. 16

[11] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, 'Non-Control-Data Attacks Are Realistic Threats.', in *USENIX security symposium*, 2005, p. 146.

[12] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, 'Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks', in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 969–986.

[13] R. S. Leon, M. Kiperberg, A. A. Leon Zabag, A. Resh, A. Algawi, and N. J. Zaidenberg, 'Hypervisor-Based White Listing of Executables', *IEEE Secur. Priv.*, vol. 17, no. 5, pp. 58–67, Sep. 2019

[14] Z. Wang and X. Jiang, 'HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity', in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 380–395.

[15] K. Lin, W. Liu, K. Zhang, H. Xia, and B. Tu, 'HyperKRP: A Kernel Runtime Security Architecture with A Tiny Hypervisor on Commodity Hardware', in *2021 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2021, pp. 1–6.

[16] J. Liu, Y. Qin, and D. Feng, 'SeRoT: A secure runtime system on trusted execution environments', in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2020, pp. 30–37.

[17] K. K. G. Buquerin, 'Security Evaluation for the Real-time Operating System VxWorks 7 for Avionic Systems', PhD Thesis, Technische Hochschule Ingolstadt, 2018.

[18] K. Müller, G. Sigl, B. Triquet, and M. Paulitsch, 'On MILS I/O Sharing Targeting Avionic Systems', in *2014 Tenth European Dependable Computing Conference*, May 2014, pp. 182–193.

[19] B. Glas *et al.*, 'Automotive safety and security integration challenges', *Automot.-Saf. Secur. 2014*, 2015

[20] A. Girmay Mesele, 'AUTOSARLang: Threat Modeling and Attack Simulation for Vehicle Cybersecurity'. 2018.

[21] S. H. VanderLeest and D. White, 'MPSoC hypervisor: The safe & secure future of avionics', in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, Sep. 2015, pp. 6B5-1-6B5-14.

[22] A. M. Nasser, D. Ma, and P. Muralidharan, 'An Approach for Building Security Resilience in AUTOSAR Based Safety Critical Systems', *J. Cyber Secur. Mobil.*, pp. 271–304, Dec. 2017

[23] P. Derrin, D. Elkaduwe, and K. Elphinstone, 'seL4 reference manual', *NICTA-Natl. Inf. Commun. Technol. Aust.*, 2006

[24] G. Heiser, 'The seL4 Microkernel–An Introduction'. The seL4 Foundation, 2020.

[25] G. Klein *et al.*, 'seL4: Formal verification of an OS kernel', in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 207–220.

[26] G. Klein *et al.*, 'Comprehensive formal verification of an OS microkernel', *ACM Trans. Comput. Syst.*, vol. 32, no. 1, pp. 1–70, Feb. 2014

[27] Z. Mi, D. Li, Z. Yang, X. Wang, and H. Chen, 'Skybridge: Fast and secure inter-process communication for microkernels', in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–15.

[28] A. Lyons, K. McLeod, H. Almatary, and G. Heiser, 'Scheduling-context capabilities: a principled, light-weight operating-system mechanism for managing time', in *Proceedings of the Thirteenth EuroSys Conference*, Porto Portugal: ACM, Apr. 2018, pp. 1–16. Accessed: Mar. 25, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3190508.3190539

[29] B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury, and G. Heiser, 'Timing Analysis of a Protected Operating System Kernel', in *2011 IEEE 32nd Real-Time Systems Symposium*, Nov. 2011, pp. 339–348.

[30] T. Sewell, F. Kam, and G. Heiser, 'Complete, High-Assurance Determination of Loop Bounds and Infeasible Paths for WCET Analysis', in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2016, pp. 1–11.

[31] D. Cofer *et al.*, 'A Formal Approach to Constructing Secure Air Vehicle Software', *Computer*, vol. 51, no. 11, pp. 14–23, Nov. 2018

[32] S. Hiergeist and G. Seifert, 'Internal redundancy in future UAV FCCs and the challenge of synchronization', in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, Sep. 2017, pp. 1–9.