Secure and Trusted Open CPS Platforms

George Kornaros
Technological Educational Institute of Crete, Greece
Ernest Wozniak
fortiss GmbH, Germany
Oliver Horst
fortiss GmbH, Germany
Nora Koch
fortiss GmbH, Germany
Christian Prehofer
fortiss GmbH, Germany
Alvise Rigo
Virtual Open Systems, France
Marcello Coppola
STMicroelectronics, France

Definition and Architecture of Open CPS

Cyber-physical systems (CPS) are devices with sensors and actuators which link the physical with the virtual world. In many application areas of CPS, like automotive or medical, devices are long lived and users depend on them in their daily lives. In the past, many of these systems have been operating unchanged for years or even decades in a well-defined context. With the rapid innovation cycles in many IT services and technologies, there is also a need to extend or update these services. For instance, functionality in cars has been used as originally shipped for the full lifetime of a car. With latest innovations in infotainment and autonomous driving, it is expected that this functionality is outdated after a few years. Thus, there is a strong trend towards open cyber-physical systems, which can be extended by instantly adding functionalities on demand.

An overview picture of such an open cyber-physical system is shown in Figure 1. This shows a network of CPS nodes, which are orchestrated to perform distributed control services and user interaction tasks. For these kind of networks, there is often a need for real-time communication, hence networks like CAN or deterministic Ethernet (TSN) are used. The presented cyber-physical system architecture envisions a single device, the Open Apps Platform device, to be the only connection point to external untrusted networks. Beside its gateway functionality, this device also provides an open platform for adding new application software (“apps”) and is the main focus of this chapter. This does not mean other devices cannot be changed or extended, it just illustrates that at least one device must be extendible. The gateway functionality of the device is needed for many applications, but also for managing the applications on the device.

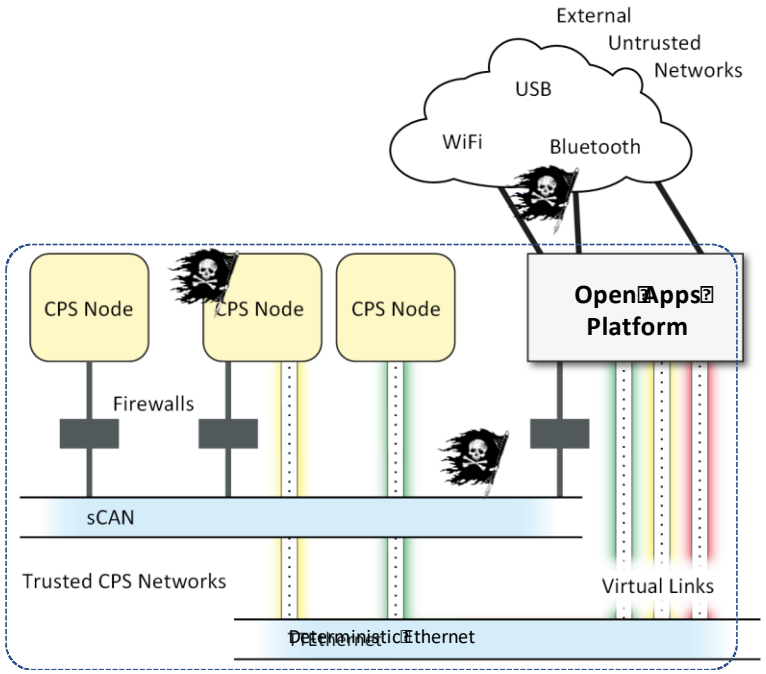

Figure 1: Open CPS Architecture

Security for such open, networked CPS systems is very important as it is a prerequisite of safe and reliable operation of the overall system. For networked CPS systems, a number of external and internal attacks can threaten the correct and safe operation of the system, as shown in Figure 1. For instance, internal CPS nodes or networks may be attacked or compromised. The specific focus of this chapter is the extension of such systems with new apps. These must not be able to interfere or compromise proper operation of the system at any time. For the new apps, there can be different levels of trust. Yet even for fully trusted software components, programming errors, software weaknesses or failures can lead to a compromised situation.

Thus, the goal of this chapter is to present the research challenges and solutions of ensuring security and safety in these new platforms for such open systems. A main problem is that such CPS apps shall be able to access and modify safety critical device internals. Thus, it is not sufficient to isolate the apps from other parts of the system, as they need to properly access and control parts of the system. This access to internals of the CPS system must however be separately controlled and monitored.

Historically, security was achieved through the isolation of subsystems and with proprietary protocols, however, as the number of interconnected systems rises, tools and software are becoming more available to malicious users, causing reverse engineering and brute force attacks to become a prominent threat to the safety and security of cars. Attacks penetrating the integrity of vehicular systems and medical devices have brought to sharp focus the urgency of securing cyber-physical systems.

Cyber-physical attacks can affect the integrity, availability and confidentiality in CPS. Examples range from deception based attacks such as false-data-injection, sensor and actuator attacks, replay attacks,

and also denial-of-service attacks. Documented defense mechanisms can range from attack identification and detection, intrusion detection as well as physical watermarking of valid control signals. Hence, new methods are required to develop an end-to-end solution for development and deployment of trusted apps. Ideas of such methods plus an architectural approach with its key components and solutions for open CPS apps are presented in this chapter. Furthermore, a possible tool chain and development support is discussed. The overall idea is to provide a layered approach, consisting of multiple, independent defense mechanisms. In conjunction with the layered architecture, different classes of applications can be defined, depending on their criticality. Then, these classes are separated and different mechanisms can be applied to them.

This chapter is organized in the following way: First, the Application Domains section discusses possible domains in which open CPS platforms may play a dominant role. Next, a set of challenges and requirements is defined for open platforms. Following is a discussion of an end-to-end solution which spans from the executions environment and apps platform up to the toolchain which supports the development of apps. Finally, future directions are sketched for open CPS platforms.

Application Domains

Open Cyber Physical Systems (CPSs) create new opportunities for value creation across the whole society and in particular enterprises will benefit from the flexibility and 3rd party's services developing and providing applications. For industries like automotive, health, utilities, transportation, home entertainment and agriculture that are increasingly using cyber physical systems, open CPSs are providing new business opportunities. For example, sensors and actuators monitoring farms, which are connected with management systems, are enabling real-time adaptive and highly efficient farming. Another example are vehicles exchanging information and apps that may improve traffic flow and parking slots assignment. These examples are just scratching the surface of the possibilities that secure and trusted apps, operating in open CPS platform context, can provide. Extensions and updates of functionality during operation using apps for CPS devices, like vehicles and medical devices, that have a long lifetime when compared to innovations in the ICT area, has the additional benefit of extending the CPSs usage.

The trend towards open CPS devices and apps-based platforms for vehicles is currently a highly active topic in the automotive domain. Traditionally, protection in this domain has been achieved by partitioning components across distributed modules, which communicate over a network such as a CAN bus. However, when applications are able to access that network, there are several ways to break the protection. Therefore, new concepts and implementations are needed at hardware and software level to enforce security and safety for the driver when using open CPSs. Here it has to be distinguished between apps that interface with external, well known Internet services and access status information of vehicles, and highly trusted apps, which can also access critical information and settings. For instance, an example for the first app category is the check of the route based on traffic conditions and battery conditions of the e-vehicle. Instead an app that changes settings in the engine control of a vehicle like brakes or gears according to environment conditions such as weather and terrain is highly critical.

Also in the healthcare market, medical devices require high-level secured architecture and trusted apps due to the increasing risk of cyberattacks. Such attacks can cause critical and physical damages if the targets are e.g. wearable and implantable CPSs, surgical robots or drug delivery systems, threatening people health and/or producing concrete economic damages to healthcare systems, especially in terms of patient safety. Although it is not possible to quantify the impact of cyberattacks on healthcare devices, there are many reports on damages already caused (Burns, Johnson, & Honeyman, 2016) It is therefore evident that healthcare devices must include strong security mechanisms to ensure, on the one hand, patients' safety, and on the other hand, privacy and security of data. Once security mechanisms can be guaranteed through a trusted platform, the medical device producers in the healthcare domain will benefit from the use of trusted apps for updating and improving functionality of their devices.

Other domains like those in the context of Industry 4.0 are addressing security and trustworthiness of open CPSs, too. New approaches and technologies are being developed or existing ones are adapted and/or extended to be used in these areas such as factory automation and cyber-physical production systems. An analysis of the requirements in the different domains regarding support for open cyber physical systems will provide the basis for the development of a domain-independent platform for trusted open CPSs.

Challenges and Requirements for Open CPS

Authors of (**Fehler! Verweisquelle konnte nicht gefunden werden.**) sketch a very futuristic vision of new applications enabled by the sensing and actuation utility and plethora of embedded devices (such as mobile phones, vehicles, etc.) connected altogether to the Internet. Cars as well as aircrafts talking to each other to avoid collisions or uploading of a physiological data to doctors in real-time with real-time feedback are examples of such vision. It is not just the applications itself that will change but also their installation and running which will become much easier. To reach these benefits, the provision of an open platform is necessary. These benefits of an open platform are however challenged with the new problems that span across many aspects.

i. For instance, definition of new communication interfaces and contracts for specifying interactions will be required, to enable the exchange of information of diverse systems cooperating within the sensing and actuation utility. This is important especially if considering the flexibility of an open platform to accept during its operation new applications which in any case might have to communicate.
ii. Deep intertwine of physical and software components characterizing CPS and potential cooperation of apps deployed on the same, open CPS platform, giving potential for alternative services, will require definition of novel control algorithms or unification of mathematical description for dynamical interactions (**Fehler! Verweisquelle konnte nicht gefunden werden.**) in order to address the new operation conditions.
iii. Significant challenge of future open CPS platforms is to allow the execution of multiple applications of different criticality, so called mixed-criticality environments. This poses an even

larger challenge for multi-core based platforms. The integration of mixed-criticality subsystems can lead to a significant and potentially unacceptable increase of engineering and certification costs if proper preconditions specification isn't put in place. For example, in order to prevent consolidated applications running on a multi-core from interfering with each other, spatial and temporal isolation of the shared resources is mandatory (Richter, Herber, Rauchfuss, Wild, & Herkersdorf, 2014).

iv. The techniques for design, integration and deployment of applications in such highly distributed ecosystem have to be considered as well. For instance, programming languages must allow inherent integration of time-based computation with event-based computation, which will enable effectively modeling asynchronous dynamics that take place at different temporal and spatial scales (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

v. Analysis techniques such as response time analysis validating real-time properties will have to be tailored or in fact, elaboration of new types of analysis will be necessary. For example, in the context of an open platform, some checks before new application is accepted for running will have to be made in order to determine whether real-time properties of already deployed apps will not be affected.

vi. Software development for CPS per se is a sophisticated endeavor which yields many challenges (**Fehler! Verweisquelle konnte nicht gefunden werden.**). Therefore, new development tools will be needed to address the new philosophy of apps development, integration and deployment and to support techniques for their analysis.

vii. In many cases, concepts of futuristic applications directly or indirectly influence safety-critical parts of the CPS devices or exchange large amount of sensitive data. Consequently, the main challenge is to ensure security and safety. Malicious activity has continued to follow advances in technology, as it can now be seen with exploitation of mobile devices and infrastructure. Unfortunately, it might be impossible to reuse some of the currently existing security solutions as they consume lots of memory and require heavyweight computational power. Hence new, lightweight algorithms might be needed to find the balance between security and for instance power consumption of devices that will employ them (Jing, Vasilakos, Wan, Lu, & Qiu, 2014).

Accordingly, when considering the highlighted challenges, a successful approach for Open CPS platform should implicitly guarantee the integrity, safety, security and observance of some non-functional properties such as real-time requirements. In that respect the main key requirements, which need to be addressed by the open CPS platform are:

i. App isolation. Spatial as well as temporal separation is required so the applications can be executed in parallel, without any undesirable influence from the outside apps. Unless intended, any influence on app data or its execution behavior should be prevented.

ii. Access control and resource management. Access to crucial resources and functions offered by the platform should be controlled. For example, entry to the critical communication interface such as the CAN bus should be restricted to apps which are certified or require real-time execution. Also proper management or safeguarding of resources through the execution platform is required. This is to prevent from possible

damage of the resource or its unlimited blocking by one app, causing violation of real-time properties of other applications which also require an access to it.

iii. Real-time support. Apps running on the CPS platform in many cases need to perform real-time computation, e.g. real-time traffic update or even live video feed of a planned route for the vehicle driver (Wan, Zhang, Zhao, Yang, & Lloret, 2014). Guarantees for real-time behavior needs to be given considering parallel execution of other applications on the same platform, which might equally be real-time oriented. For this, platform should offer real-time support, e.g. via usage of a real-time operating system, whereas development tools can offer analysis techniques (e.g. schedulability analysis) to verify observance of real-time properties even before the deployment.

iv. Reliability regimes. CPS platform should support several reliability regimes for safe and secure operation of safety critical applications. Modes such as fail-operational, fail-safe or fail-secure fall within this group.

v. Privacy policies. Apps deployed on the platform might have to include specification of privacy policies so that the requests handled by the app can be first evaluated in respect to them.

vi. White-box app development. Certification of critical applications might impose a rule for white-box development to give an insight of the certification authority into the developed sources.

Further advancements in the security and safety related challenges can be accomplished by incorporating multiple safety/security layers, spanning from the trusted hardware, through computing and network virtualization, communication middleware, up to the point in which apps are developed. Usage of specific technologies might be required to support such multiple lines of defense constituting an end-to-end approach. These technologies are described in the following section.

End-to-End Solution

In the next generation of open cyber-physical systems where malfunctions could cost lives, security concerns include several different points of view: communication, system and software architecture. To guarantee a trusted app infrastructure it is mandatory to provide multiple safety-security layers to counter attacks and prevent malicious behavior. There are several technologies identified as imperative installments into a multi-layered defense strategy of an open CPS. On the hardware level, mechanism such as ARM TrustZone (ARM Limited, 2009) might be used in order to separate different execution environments. Other HW solutions can be employed for further separation of execution environments. These are memory management units or specific on-chip networks. Another technique this is Virtualization, used to provide virtual execution environments, isolation and protection of resources and spatial plus temporal isolation of apps. Next is communication middleware for securing and controlling the inter app communication as well as controlling an access to different resources. Lastly the support for the safety and security concept can be achieved by delivering specific toolchain. The development of critical applications might be restricted to the usage of such trusted toolchain. Via incorporation of some analysis and verification techniques, a toolchain might certify safe and secure operation of application (observance of non-functional constraints, correct execution behavior) even before the deployment. Finally, the critical apps developed with provided toolchain can be checked by trusted third-party that will issue a certificate acknowledging their safe and secure behavior. Hence the partially white-box

development approach will be incorporated. Partially, because only the certification authority will be eligible to inspect the source code.

Trusted apps platform needs to ensure strong isolation and real-time properties for some apps. This cannot be assured by existing non-real-time operating systems. Moreover, existing platforms are characterized by high complexity and support for many different APIs which is a potential loophole for security threats. It is easier to assure safety/security of operation when considering rather more limited operating system. On the other side, to attract app developers it is essential that the platform offers features, such as feature-rich developer support, or simply compatibility with existing platforms and tools. This opens up a possibility for reusing currently existing tools, already developed applications or simply developers' knowledge. These contradictory concerns pose an interesting research challenge for developing execution platform that can provide multiple execution environment (EEs) intended for apps of different types and needs. Applications with a lower criticality can be run on systems like Linux or Android but enhanced with protection mechanism discussed above. High criticality apps can on the other hand be executed on the smaller, real-time operating system. This of course means less support in terms of APIs but on the other side it can assure more timely behavior of safety-critical and real-time use-cases.

Execution Environments and Apps Platform

As mentioned earlier in this chapter, it is of crucial importance that Open CPS platforms are able to address security and safety requirements. These have to be all sustained by the CPS architecture, which is comprehensive of the software stack, of hardware components and related extensions. Hardware manufacturers are now proposing an increasing number of SoCs that fit the Open CPS platforms use case, in such a way to support real-time execution and strong isolation for non-interfering applications of mixed levels of criticality. For instance, ARMv8 processors are examples of commonly used CPUs that include, in one package, virtualization and TrustZone extensions, two technologies presented in the "End-to-End Security" Section combination of which can have interesting applications in the context of Open CPSs, as will be discussed in the following lines.

The use of virtualization for instantiating Virtual Machines is nowadays a widely used technique, which allows to flexibly deploy almost any operating system, the guest OS, on top the host OS, offering to the former an isolated partition in which to execute. The host OS is usually a general-purpose OS like Linux which, together with KVM, offers a solid virtualization infrastructure used in millions of servers out there, sustaining most of the cloud services running nowadays.

Considering an OS the only possible "guest" of a Virtual Machine is too limiting: virtualization can also be used to secure single applications that run alone inside a Virtual Machine, on top of a very simple OS (Kivity et al., 2014). In essence, a Virtual Machine can either be a self-sustained system that serves one or more functions, or it can be as well a component in a disaggregated architecture which exploits virtualization to distribute applications and services to different Virtual Machines according to some

design choices. In CPS contexts, virtualization can therefore find a viable solution for running non-interfering applications in one unique platform, by confining to different execution environments the applications of the CPS. As a matter of fact, the high degree of isolation brought by virtualization comes with a cost. In fact, the more Virtual Machines are running in the system, the higher is the emulation overhead, which translates to higher resources usage of the host OS.

Clustering applications to Virtual Machines according to their criticality can be the best compromise between isolation and overhead: an application running in one Virtual Machine will be closer to applications of similar criticality, allowing for faster communication with its siblings. On the other hand, two applications of different criticality will necessarily run in different Virtual Machines, with a more complex communication mechanism interposed between them. With no surprise, such a communication introduces additional overhead: no matter what protocol will be used to allow information passing between the Virtual Machines, the communication will necessarily require at some point a device to be used, involving additional emulation. Luckily, modern virtualization techniques are able to alleviate emulation overhead, consequently lowering communication costs. As an example, paravirtualized devices, using at their own advantage the awareness of running on a virtual environment, permit to shorten the long code paths that are typically required by a faithful emulation of real devices (Russel, 2008). Low-latency communication is also possible between Virtual Machines. In fact, the implementation of ad-hoc devices and corresponding drivers for the guest OSes allow to have performance comparable to shared memory-based communication mechanisms (Macdonell et al., 2011). In essence, virtualization is now able to offer high isolation at a relatively low price, assuming that state-of-the-art solutions are employed and tuned to the specific use case.

Virtualization can therefore be used to instantiate corresponding Virtual Machines, each of them, to one “execution environment”, which hosts services and applications of a given criticality. For everything which is not safety critical, an OS like Linux, Windows or Android can fit and be deployed in these execution environments; this would allow to cover use cases from the execution of the IVI system in a car to the provisioning of web services in a connected device.

Though, the architecture depicted so far does not fit the use cases of CPS executing applications that have to obey to safety-critical requirements. Such applications come often with real-time constraints that have to be fulfilled as well. Moreover, in the most demanding cases, those safety-critical applications require also to be certified according to some safety standard (for instance, ISO 26262 in the automotive domain, or the EN 50128 for the railway systems). In these cases, general-purpose OSes

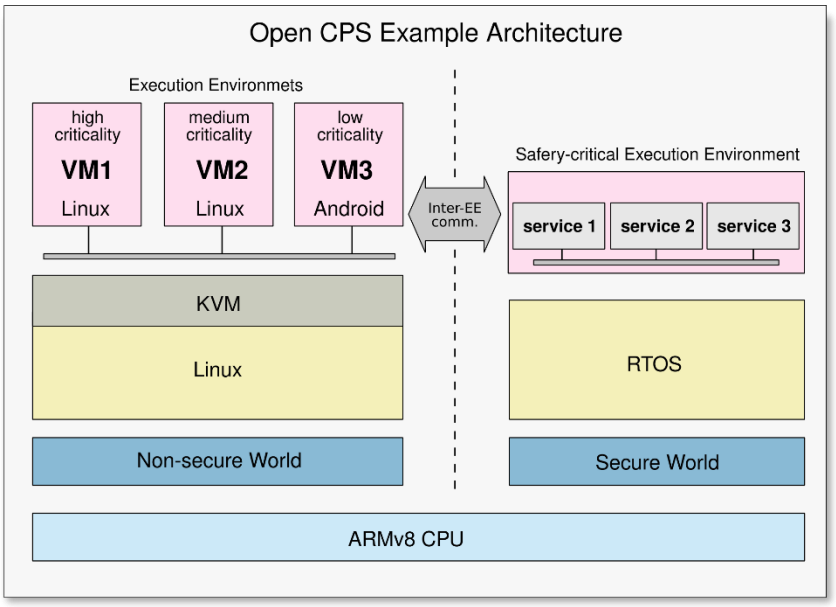

Figure 2: Example of Open CPS architecture based on ARMv8 architecture

can't be considered as viable solutions, in that they are not real-time, nor certified.

It has been demonstrated (Prehofer et al., 2016) that the ARM Trust-Zone extension can be used to run simultaneously in the same CPU two different OSes, a general-purpose and a real-time one. The so-called Secure side in the CPU, which is normally used to implement secure services, is instead used as isolated compartment to run the real-time OS, adding in this manner a new execution environment for safety-critical applications and services. By using in a clever way the capabilities of the ARMv8 architecture, it is possible to guarantee that the real-time OS is periodically and deterministically scheduled, if needed at the expenses of the general-purpose OS which has lower priority. The safety critical execution environment keeps nevertheless its secure connotation since it runs in a secure segment of the system RAM and uses secure devices that are not shared with the general-purpose OS. This makes the safety-critical execution environment ideal to host trusted services (encryption, decryption, fingerprint validation, etc.), addressing once more another requisite of modern CPS systems.

Secure and Real-time Resource Management

One big challenge of future Open CPSs is to allow the execution of multiple applications of different criticality, so called mixed-criticality environments. Anderson, Baruah, and Brandenburg (2009) and Mollison, Erickson, Anderson, Baruah, and Scoredos (2010) presented that this poses an even larger challenge for multi-core based platforms. In mixed-criticality environments it is of special interest to manage the information flow, i.e., communication, between low-critical and high-critical apps, as otherwise provided information from a low-critical app could compromise a receiving high-critical task (Sha, 2009). Furthermore, Pellizzoni et al. (2009) demonstrated that integrated mixed-criticality systems can be made safer by enforcing application specific constraints. Guaranteeing a safe integration of mixed-criticality tasks on one platform is also a matter of guaranteeing a proper real-time behavior for critical apps, even though low-critical apps are executed in parallel. Determining valuable worst case execution bounds for applications on multi-core platforms, however, is an own research challenge (Jacobs, 2013; Mushtaq, Al-Ars, & Bertels, 2013). The main reason for that are implicitly (e.g., buses) and explicitly (e.g., memory) shared resources. Kotaba, Nowotsch, Paulitsch, Petters, and Theiling (2013) provide a comprehensive list of shared resources on contemporary multi-core platforms in conjunction with the possible interferences that can change the timing of resource accesses.

Many researchers focused on a variety of solutions for the issue of sharing resources in mixed criticality systems. Solutions range from bus arbitering (Hassan & Patel, 2016) and bounded interference approaches (Nowotsch et al., 2014) for interconnects, to resource servers as dedicated resource managers (Brandenburg, 2014) and the isolation of all critical tasks to a higher prioritized core (Ecco, Tobuschat, Saidi, & Ernst, 2014).

The authors consider the secure and reliable resource management as one curial aspect of future Open CPS platforms. Accordingly, the authors studied how to integrate such management structures into the Open CPS platform discussed before. For that reason, the Safety Integration Layer (SIL) was developed. The SIL is a pivotal component that ensures a safe behavior of the overall system; it shall perform tree main actions:

1. Protect critical platform interfaces/APIs.
2. Guarantee the overall system integrity and the assured service levels for apps.
3. Guarantee a reliable real-time behavior for the communication among apps, as well as their resource accesses.

Figure 4 illustrates the placement of the SIL and its components with respect to the previously presented platform for Open CPS.

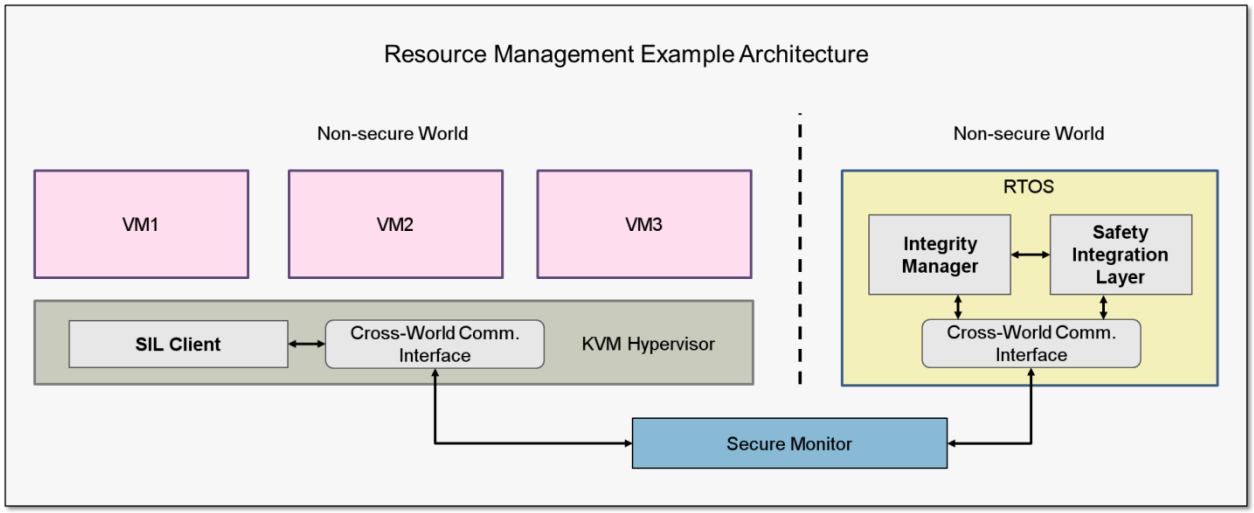

Figure 3: Example of a possible integration of a global real-time capable resource management in commodity architectures based on the ARM TrustZone technology.

It can be seen that the SIL concept introduces three new components. Beside the Safety Integration Layer itself, these are the Integrity Manager and the SIL Client. The Integrity Manager shall ensure that all critical apps can access there declared resources at any time without interference by other apps and in accordance to their timing requirements. This includes schedulability tests during the deployment of new applications, as well as calculating new schedules for the different resources of the platform that conform with all requested real-time behaviors of the installed apps. In case no feasible schedule can be found the Integrity Manager can also reject an app during the deployment process. The SIL Client, on the other hand, extends the control range of the SIL to the non-secure world. As part of the KVM hypervisor, the SIL client moderates the resource accesses of the various virtual machines on behalf of the SIL. Communication between execution environments in the two worlds, as well as between the SIL and its client is done via the cross-world communication interface, which uses the Secure Monitor of the ARM TrustZone technology to switch worlds and safely exchange data between the two worlds.

To guarantee the overall system integrity, the assured service levels, and a reliable real-time behaviour, the SIL and its client in the KVM hypervisor coordinate the communication and resource accesses of all apps with a globally enforced schedule. The Integrity Manager ensures that there is always a feasible schedule. Furthermore, the SIL creates isolation domains in hardware and software. On the one hand, it partitions the hardware (i.e., memory and interconnects), and on the other hand, it isolates apps in their own software fault domain. The latter is done with the help of Software Fault Isolation (SFI) techniques (Ruhland, Prehofer, & Horst, 2016). The SFI technology isolates even faulty apps within their own fault domain, and specifically controls the utilized instruction set and calls to the underlying platform within each app. The SIL then operates on these defined APIs and enforces the requested service levels per app.

Due to its supervising characteristic and the focus on resource management, as well as communication, the authors propose to implement the SIL as an extension to an existing middleware. This middleware guarantees real-time constraints for the communication on the platform by globally coordinating bus

and memory accesses. Additional resource managing components in the SIL are responsible for implementing a globally coordinated schedule on the resources. Thus, each request to a resource is supervised by the SIL, either in the sense of a middleware, or a resource arbiter.

Secure and Trusted CPS Networking

Cyber-security-safety issues are undoubtedly crucial for CPSs since the entities within these systems interact not only with each other, but also interact with the environment. CPSs integrate embedded computing devices, physical processes and networks, which makes it necessary to develop safety protection mechanisms at multiple layers beyond computing, in order to consider distributed networked devices. Modern CPSs present today an increased attack surface, due to its networking and coordination functionalities, due to the integration of larger number of interoperable devices (such as medical devices), and most importantly due to increased software components and open-ness (such as modern automobiles, vehicle-to-vehicle (V2V) and machine-to-macine (M2M) communication trends) (Weimerskirch, 2014). Safeguard infrastructures need to detect and subsequently prevent threats and vulnerabilities of open CPSs that may cause corruption of sensor and control information and disruption of the physical system, or exposure of confidential information. In addition to data security, aggressors may initiate attacks that target the real-time properties of a CPS. Real-time accessibility gives a stricter operational environment than conventional CPSs.

Major concerns in CPS communication include keeping the data private and allowing only authorized access. Attackers may not only attempt to physically probe the devices, altering their behavior or intercepting the physical properties of power consumption and timing behaviors to analyze the secrets and masquerade them, but can also implement network intrusion at the physical layer as well as the software layer. All relevant safety standards assume that a system's usage context is completely known and understood at development time. This assumption is no longer true for open CPSs, meaning that their security vulnerabilities could be faults leading to life-endangering safety hazards. As many CPSs are becoming open systems, they are the target of cyber-attacks. Interconnectivity removes boundaries and the need for physical presence to gain access. For instance, automotive architectures today include features that involve both remote diagnosis and maintenance functionality. Such systems enable Over-the-Air (OTA) updates and the management of vehicle functions over communication links. As many diverse communication networks are integrated, gateways are employed to connect different parts and enable functions to obtain needed information from all parts of the vehicle. Since gateways are able to access all buses and hence connected devices, updates or remote access functions are mostly supported in such components.

Modern vehicles support WiFi, Bluetooth connectivity, many diverse CAN bus systems, dedicated networks for advanced driver assistance systems (ADAS) and may connect even up to 100 ECU modules. In another domain, the remote medical-support services or tele-assistance systems trust of personal devices as well as trust in transmitted information is crucial. Multiple dimensions of trust are important, such as (i) the safety of using the devices in tele-presence spaces (e.g., safety of blood-pressure devices attached to a user or safety of a sensory instrument), (ii) reliable and timely information delivery, (iii) stability of the overall system, (iv) low risk in receiving wrong information, and (v) privacy guarantees. Current systems still do not have this level of sophistication, and an appropriate trust configuration remains a challenge; especially since the verification and validation of a cyber-physical-system is not a one-time event. Instead, intra- and inter- CPS communications to support not only internal connectivity but also over-the-air connectivity, it is essentially a life-cycle process continuously ensuring the

certification of safety critical services. In addition to trusted platform modules (TPMs) that can be used to enable trusted boot, where each piece of code loaded from boot-time is measured via cryptographic hash before loading, CPS communications need to be secure, taking measures to ensure trusted communication from both active (interferers) and passive (eavesdroppers) adversaries. Therefore, to guarantee secure and trusted networking in these emerging CPSs, methods are developed towards enhancing network protocols in terms of security and towards designing CPS devices with security in mind.

Network-level Support Detection and Prevention

The networking of gateways, switches, and firewalls components for wired, wireless and sensor networks in CPSs should ensure routing security, traffic control on information flow, and the necessary network separation (demilitarized zones (DMZs). Moreover, in order to fulfill authentication requirements, developers propose more efficient schemes during the design or upgrade of communication protocols (Wang et al, 2009). A CPS network protocol should ensure that a transmitted message is authentic and determine if the integrity of the message has been compromised. An authentic message indicates that the device alleged to be the sender of the message is actually the device that sent the message. Message integrity denotes that the contents of the message received by a recipient device have not been altered after having been sent from the alleged sender. Cyber threats can involve forging communication messages that appear to be from a trusted sender, but that are not actually from the sender, or eavesdropping on legitimate messages from the sender and attempt to spoof the receiver with copies of the legitimate messages. Detection and prevention protocols must prevent the attacker from convincing the recipient that the message is from a trusted device or that the contents of a copied message can be trusted.

Previous works employ behavior-based techniques for intrusion detection (Sun et al., 2008), optionally using domain-specific knowledge and are often targeted at wireless communication. Protocol extensions towards intrusion prevention protocol can utilize different key establishment with regard to the cases of deployment of networks and establishes different types of keys according to the role of a sensor node. The prevention protocol also enables to encrypt a message selectively or to append a message authentication code to its related critical proprietary information of CPSs applications.

Gateways, switches, firewalls and components are critical for cyber-security, as they can contribute to the necessary network separation. The networking of these components for wired, wireless and sensor networks should ensure routing security and improved resiliency against cross-layer traffic injection as specified by National Institute of Standards and Technology (2014). In modern and future vehicles for example, to address increasing complexity due to the large number of Electronic Control Units (ECUs) and their software code, a promising solution is domain-based networking. In such a set-up a domain controller isolates a number of devices (cost reduced “light” ECU) using for instance standardized Software (AUTOSAR). From a threat perspective, communication segmentation both on-chip and off-chip allows better visibility, management and isolation. Essentially, as Figure 3 shows an automotive example, the division of the network into security zones advances monitoring of internal traffic and devices, prevents unauthorized access to restricted data and resources, and controls the spread of intruders and malware, as well as error propagation.

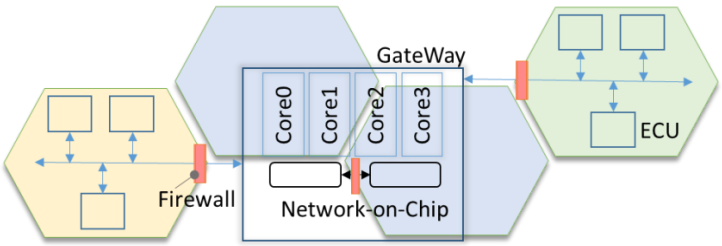

Figure 4. Example of an automotive networked CPS with various Electronic Control Units (ECUs) supporting segmented on- and off-chip networks.

CPS Device-level Support for Detection and Prevention

In addition to ensuring secure communication, modern open CPSs also involve hardware primitives for security and authenticating the CPS device itself and mechanism to ensure its own underlying binary code is trusted. ARM's TrustZone Security Extensions discussed by ARM Limited. (2009), enable processor and memory isolation effectively creating two distinct "worlds"—the secure world for security sensitive application code and the normal world for non-secure applications. Hardware solutions also offer support in on-chip networks for logically isolated multi-compartments (Kornaros et al., 2015) and hardware monitors for secure embedded devices (Mao and Wolf, 2010), or hardware support for virtualization of I/O devices such as the CAN controller in automotive and extensions that guarantee a spatial and temporal isolation of virtual controllers (Herber et al., 2014). To ensure trusted communication, proposals include for instance methods of integrating physically unclonable functions (PUFs) (A PUF is a complex physical system with a large number of inputs and outputs, where the mapping from the inputs to the outputs cannot be predicted in any reasonable time, and the system cannot be reproduced due to scientific or technological difficulties) along with existing hardware in the design to create a trusted information flow within the system (Potkonjak et al., 2010). Digital PUFs have been designed for enabling remote secret key exchange protocol and communication tasks, because both communicating parties experience very low overhead in terms of both time and energy.

Finally, software methodologies can provide enhanced security in deeply embedded real-time CPS systems. Various methods employ application instrumentation to detect anomalies, such as timing dilations exceeding worst-case execution time (WCET) bounds in order to attain elevated security assurance. Static timing analysis is performed on selected code sections to obtain bounds during the right timing for the required schedulability analysis, and the bounds are subsequently utilized to monitor execution during run-time (Zimmer et al., 2015). Intrusion Detection Systems (IDS) are also widely deployed to detect unwanted entities into a system by using signature-based, specification- based, or anomaly-based techniques. To achieve vehicular bus communication security authentication of all senders in the gateways, encryption of data transmissions and network firewalling are fundamental techniques (Kleberger et al., 2011).

Existing techniques for securing CPS networking have various drawbacks, especially in broadcast and multicast systems with limited computing power and data storage. However, the vast majority of cyber-attacks can be prevented through optimizing communication standards, incorporating hardware security attributes in device designs, upgrading firmware to support trusted root of trust, etc. By considering the implications of intercepted, deleted, modified and forged information from all components of a networked CPS, designers increasingly adopt the appropriate steps to protect the system end-to-end, against such attacks.

Secure and Trusted Inter-EE and Inter-App Communication

The idea that the execution environments are characterized by a priority or criticality has not to prevent them from communicating with each other. The information sharing between heterogeneous contexts of execution is one of the key feature to implement functional and full featured CPS systems. However, the trustworthiness of this communication, both between apps running on the same execution environment (Inter-App) and running on difference execution environments (Inter-EE), is of pivotal importance for the safety and security of the entire architecture. For this reason, the techniques that are used by apps to communicate need to be supervised by software, as well as by hardware. In the remaining part of this section, a review of the available techniques that allow communication between applications, virtual machines and, more generally, execution environments will be given, proposing as well ideas for novel approaches.

The communication between Virtual Machines is a topic that has been vastly explored in the past, since it introduced room for improving the canonical communication based on physical network cards and physical switches. The emerging paradigm of Software Defined Network (SDN) and the subsequent creation of the Network Function Virtualization (NFV), made this interest even stronger. The concept of NFV is about decoupling the software implementation of network functions from the underlying hardware (Azodolmolky et al., 2013), confining the former in Virtual Machines. All this attention allowed to burst significantly the development towards highly efficient and fast solutions aimed at switching network packages between Virtual Machines. Relaying on the Linux subsystems, it was already possible to implement decent switching mechanisms between VMs' network interfaces. In fact, assigning one TAP device to every VM it is possible to benefit from the in-kernel packet switching, technique which is normally used when Linux serves as a router/switch OS. This is not an ideal solution in the context of virtualization, since the handling of incoming/outgoing packets require expensive exits from the VM which will eventually bring the execution to kernel space, where the actual packet switching happens. The packets passing though the in-kernel switching modules can be, therefore, filtered by Linux modules such as netfilter, the Linux packets filtering solution. This, together with iptables, offers a full-featured firewall solution. Other types of packets analysis (e.g.: packets inspection) are difficult to achieve in these situations from a regular user space application for security reasons.

Avoiding the commutation to kernel mode during the packet switching would bring performance and latency improvements; this is the starting point of several virtual switches that have been implemented in the past and that are still heavily used as virtual functions. Example of virtual switches are VOSYSwitch (Paolino et al., 2016), Snabb, VALE (Rizzo et al., 2012) and OVS-DPDK: all these solutions implement in user space the switching functionality, reducing the costs associated to virtualization, requiring less context switches. Flexibility-wise, these solutions are much more convenient than the in-kernel solutions: project like VOSYSwitch provides the API and the needed infrastructure to enrich the virtual switch with plug-ins, it also provides extended functionalities like virtual LANs, firewall, packets

inspection and so on. Such a flexibility finds application in the CPS architecture, enforcing security policies between VMs.

The virtual switches, although being ideal solutions as communication between VMs (and so to non-safety-critical execution environments), do not address the communication between an execution environment in the Normal-world, and the RTOS (or safety-critical execution environment). The hardware partitioning of the two worlds makes impossible to use conventional communications methods: the link between a process running on the Secure world and the Normal world requires an ad-hoc solution. Offering the insights for such a solution is the GlobalPlatform (https://www.globalplatform.org/specifications.asp) program, which aims at standardize the interoperation between multiple applications on secure chips. Specifically, the GlobalPlatform API were designed to standardize the communication protocol between these applications, called the Trusted Applications (TA), and the user (client applications running in the Non-secure side). The Secure side, according to the GlobalPlatform naming, is called Trusted Execution Environment (TEE). These API suit particularly well the Open CPS use case, giving a ready-to-use design for the cross-worlds communication. Every application which wants to initiate a trusted communication with a TA, has to first initialize the context of execution with the desired service running in the TEE (TEE Server). Such a context provides all the necessary resources to properly execute the TA, configuring for instance the shared memory between the two processes if needed. Figure 4 depicts the overall scheme of communication between execution environments.

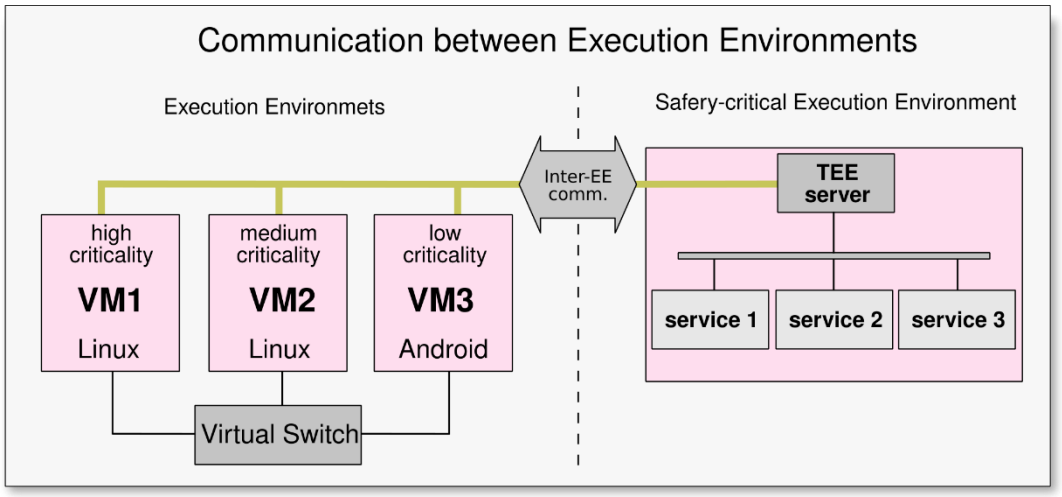

Figure 5: Communication scheme between Execution Environments: Inter-EE refers to the communication between the Safety-critical execution environments and the others. The communication between the non-safety-critical execution environments is represented by the Virtual Switch interconnection.

GlobalPlatform does not address only the communication part, but also covers the characterization of the applications running on the Secure side as well as their properties. This gives valuable guidelines for the development of the Secure eservices.

Overall, GlobalPlatform offers the technology to exploit the ARMv8 TrustZone extension towards secure communication between execution environments, ensuring the integrity of the Secure services and the confidentiality of the Secure assets.

Other solutions have been explored that, relying on TrustZone, provide a framework to implement a secure communication channels between the TEE and the Normal world. Jang et al. (2015) presented a framework which, although not following the GlobalPlatform design, provides a mechanism of secure sessions, where the TEE provides the session key to the requestor only if its code and control flow have been successfully verified.

The secure provision of a key to a process running in the Non-secure side introduces powerful possibilities, like the initialization of a crypted channel between non-safety critical processes running in the same or different execution environments. In this case, it is of fundamental importance that the session key is treated with the due measures to not jeopardize the security of the system. In fact, an improper handling of such keys can greatly extend the attack surface for malicious software (like, for instance, copying the key to non-secure memory).

Model-based Toolchain

Provision of an open platform should go in pair with the specification of an approach for developing applications and/or recommendations of tools that can be used to develop applications intended to run on such platform. Most common and a natural choice is a usage of programming languages, such as C or C++. The fact that these are the languages well known among the software engineers increases the chances of building potentially large community of developers. Indeed, the last is one of the most important objectives of open platform hence technologies which are advertised should be well recognized. Nevertheless, in the area of CPS development, there is a trend of going towards model based design (MBD) that supports the construction of abstract models and the transformation into concrete implementations. MBD is a further step in the direction of increasing the abstraction levels in the development process, offering more high-level, reusable and maintainable software components, with a main goal to speed up an entire development process, improving at the same time overall software quality as models might be a subject to formal analysis. Evaluation of properties through formal checks has a meaningful impact on the security. At the code level, software quality can be improved by unit testing, complex integration process and checks or usage of a coding standards. These are the ways to proceed when delivering the code supposed to provide certain level of quality. When moving with the development to the modeling level, additional techniques, such as model checking can be employed to check that formal defined properties of a model hold. Consequently, it is of a high interest to have a toolchain which incorporates support for programming but also exceeds the functionality of an ordinary, programming tool with the model-based design in which models are not just an artefact for code generation but also an input for verification via model checking.

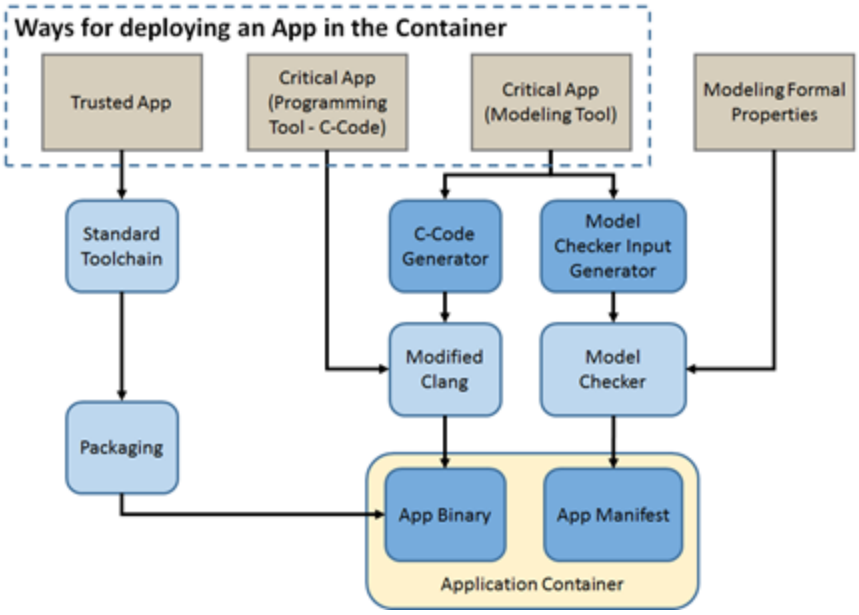

Figure 6 - Components of apps verification and development toolchain

The approach presented in this chapter tries to respond to such demand by providing a toolchain in which trusted apps can be developed and packaged using standard ARM toolchain and posted in the container, whereas critical apps can either be written entirely in native languages, i.e. C/C++, or can be modeled using tools such as 4DIAC modeling tool (*4DIAC, Eclipse Incubation Project, 2016*). 4DIAC is an IEC 61499 (*IEC 61499, 2015*) compliant integrated development environment (IDE) that provides an engineering environment to model distributed control applications. The development approach in 4DIAC follows the application centric design approaches as of IEC 61499 based systems. Overall systems are created by modeling the required applications. Out of the 4DIAC model machine deployable C code can be then generated. It is believed that code generated from models makes it harder to insert code-level attacks compared to approaches which permit arbitrary executables on a platform. Thus there are three basic ways in which apps can be deployed in the app container, all summarized in the Figure 6. The app container itself apart from the App Binary might include as well the App Manifest. The last might be for instance formatted in JSON, and for critical apps it can store special, additional information such as:

- Signed certificate, which was created by an entity that can guarantee the safety of the app. Consequently, only certified apps may be permitted to run within the Critical Execution Environment (CEE) which eliminates situations in which end user downloads faulty or malicious software.
- Resources required by the app, for example memory, exclusive hardware access and required permanent space.
- Application manifest with Plug & Play information.

Having such information the toolchain should be able to limit and analyze the features of an app, even before deployment and on the source level, without having to disclose the source code.

Furthermore, to ensure higher level of trust, security and safe operation, critical apps developed using a modeling tool can be verified if they are conforming to the platform API requirements and constraints. For instance, an app behavior in 4DIAC is specified using asynchronously interacting state-machines. Given a finite model of an application, it is now possible to systematically check if a formal property holds for the model, by using model checking technique. Example of a property that could be model checked for a vehicle app responsible for cruise control (CC) functionality, is that it will never operate for the vehicle speed lower than 15km/h. For that an existing model checking tool can be integrated. For example, NuSMV(Cimatti et al., 2002) has an open architecture for model checking and is reliable to be used in verifying industrial designs. The NUSMV project aims at the development of a state-of-the-art symbolic model checker, designed to be applicable in technology transfer projects: it is a well-structured, open, flexible and documented platform for model checking, and is robust and close to industrial systems standards. Seamless integration of the NuSMV checker can be done through the generation of the NuSMV input language (Cimatti & Roveri, 1998) (which is essentially the same as the CMU SMV input language (Clarke, McMillan, Campos, & Hartonas-Garmhausen, 1996)) out of the 4DIAC model (as shown on the Figure 6) or any other kind of modeling language used or tailored for designing CPS applications. Therefore, only apps modeled with 4DIAC can be verified. The results of the model checker are then bound with the app binary as a manifest of the application container.

Provision of mentioned features (e.g. support for formal verification) within the toolchain adds to the desire of securing the operation of apps within the open CPS platform context. There are however further ways in which the toolchain will support that concept. Guidelines for verification of a common security threats might be one of them.

Future Directions

The market of open CPSs is rapidly growing and evolving, at the same time security and trust are getting more and more relevant. Although many standard IT security concepts can be applied to open CPSs, dynamic, run-time update of critical devices should be considered explicitly in the deployment of open systems, while addressing security concerns for such devices at the same time.

The main domains – aerospace, automotive, healthcare, industrial automation – are analyzing and providing hardware and software-based technologies for addressing these security issues in their areas. But the whole CPS industry would benefit from more general domain-independent platforms and development processes, which would allow for building industrial devices and developing software services with security requirements in mind from the first step on. For example, introducing verification and model-driven development techniques such as those outlined in the previous section, in order to detect the possibility of an execution of unauthorized instructions in real-time CPS environments. By introducing early warning systems should raise intrusion detection capabilities but also might provide several steps of reduced functionality (Zimmer, Bhat, Mueller, & Mohan, 2015).

Security professionals will play a large role in the development process of open CPSs evaluating risks for organizations and deciding about solutions and tools to be applied in each individual case. Security

vendors and cyber physical system vendors should cooperate for a better detection of threats and mitigation of successful attacks. Lessons learned in other industries such as personal computers, tablets and smartphones producers should be analyzed and applied in the CPS area alike to build software security on top of hardware security modules and secure communication. CPS features like real-time, distributed components and loss of physical devices should not provide any leak to attackers. But this is not enough, security of CPS needs to be built into the design of the system itself (Moholkar, 2014).

Acknowledgments

The research leading to these results has received funding from the European Union (EU) Horizon 2020 project TAPPS (Trusted Apps for open CPSs) under RIA grant n° 645119.

References

4DIAC, Eclipse Incubation Project. (2016). Retrieved from https://eclipse.org/4diac

Anderson, J., Baruah, S., & Brandenburg, B. (2009). Multicore operating-system support for mixed criticality. In Proceedings of the workshop on mixed criticality: Roadmap to evolving UAV certification. San Francisco, CA, USA.

ARM Limited. (2009, April). ARM security technology: Building a secure system using TrustZone technology (techreport No. PRD29-GENC-009492C). Author.

Azodolmolky, S., Wieder, P., & Yahyapour, R. (2013). Cloud computing networking: challenges and opportunities for innovations. IEEE Communications Magazine, 51(7), 54–62.

Bogdan, P. (2015). A cyber-physical systems approach to personalized medicine: Challenges and opportunities for NoC-based multicore platforms. In Proceedings of the design, automation & test in europe conference & exhibition (pp. 253–258). San Jose, CA, USA: EDA Consortium.

Brandenburg, B. B. (2014, December). A synchronous IPC protocol for predictable access to shared resources in mixed-criticality systems. In IEEE real-time systems symposium (pp. 196–206).

Burns, A. J., Johnson, M. E., & Honeyman, P. (2016, October). A brief chronology of medical device security. Communications of the ACM, 59(10), 66–72.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., . . . Tacchella, A. (2002, July). NuSMV 2: An opensource tool for symbolic model checking. In E. Brinksma & K. G. Larsen (Eds.), Computer aided verification: 14th international conference, Proceedings (pp. 359–364). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cimatti, A., & Roveri, M. (1998, December). NuSMV 1.0: User manual (Tech. Rep. Nos. Technical report, ITC-IRST, Trento, Italy.)

Clarke, E., McMillan, K., Campos, S., & Hartonas-Garmhausen, V. (1996). Symbolic model checking. In R. Alur & T. A. Henzinger (Eds.), Computer aided verification: 8th international conference, cav '96 new brunswick, nj, usa, july 31– august 3, 1996 proceedings (pp. 419–422). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ecco, L., Tobuschat, S., Saidi, S., & Ernst, R. (2014, August). A mixed critical memory controller using bank privatization and fixed priority scheduling. In IEEE 20th international conference on embedded and real-time computing systems and applications (pp. 1–10).

Hassan, M., & Patel, H. (2016, April). Criticality- and requirement-aware bus arbitration for multi-core mixed criticality systems. In IEEE real-time and embedded technology and applications symposium (RTAS) (pp. 1–11).

Herber, C., Richter, A., Rauchfuss, H., & Herkersdorf, A. (2014, September). Spatial and temporal isolation of virtual CAN controllers. SIGBED Rev., 11(2), 19–26.

Huang, S., Zhou, C.-J., Yang, S.-H., & Qin, Y.-Q. (2015). Cyber-physical system security for networked industrial processes. International Journal of Automation and Computing, 12(6), 567–578.

IEC 61499. (2015). Retrieved from http://www.iec61499.de/

Jacobs, M. (2013). Improving the precision of approximations in WCET analysis for multi-core processors. In Proceedings of the 7th junior researcher workshop on real-time computing (pp. 1–4).

Jang, J. S., Kong, S., Kim, M., Kim, D., & Kang, B. B. (2015). SeCReT: Secure channel between rich execution environment and trusted execution environment. In NDSS.

Jing, Q., Vasilakos, A. V., Wan, J., Lu, J., & Qiu, D. (2014). Security of the internet of things: perspectives and challenges. Wireless Networks, 20(8), 2481–2501.

Khaitan, S. K., & McCalley, J. D. (2015, June). Design techniques and applications of cyberphysical systems: A survey. IEEE Systems Journal, 9(2), 350-365.

Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D., & Zolotarov, V. (2014). OSv—optimizing the operating system for virtual machines. In USENIX annual technical conference (USENIX ATC) (pp. 61–72).

Kleberger, P., Olovsson, T., & Jonsson, E. (2011, June). Security aspects of the in-vehicle network in the connected car. In IEEE intelligent vehicles symposium (IV) (pp. 528–533).

Kornaros, G., Christoforakis, I., Tomoutzoglou, O., Bakoyiannis, D., Vazakopoulou, K., Grammatikakis, M., & Papagrigoriou, A. (2015, August). Hardware support for cost-effective system-level protection in multi-core SoCs. In Euromicro conference on digital system design (pp. 41–48).

Kotaba, O., Nowotsch, J., Paulitsch, M., Petters, S. M., & Theiling, H. (2013). Muticore in real-time systems - temporal isolation challenges due to shared resources. In Proceedings of the workshop on industry-driven approaches for cost-effective certification of safety-critical, mixed-criticality systems (WICERT).

Macdonell, C., Ke, X., Gordon, A. W., & Lu, P. (2011). Low-latency, high-bandwidth use cases for nahanni/ivshmem. In Kvm forum (Vol. 2011).

Mao, S., & Wolf, T. (2010, June). Hardware support for secure processing in embedded systems. IEEE Transactions on Computers, 59(6), 847–854.
Moholkar, A. V. (2014, July). Security for cyber-physical systems. International Journal of Computing and Technology (JCAT), 1(6).

Mollison, M. S., Erickson, J. P., Anderson, J. H., Baruah, S. K., & Scoredos, J. A. (2010, June). Mixed-criticality real-time scheduling for multicore systems. In 10th IEEE international conference on computer and information technology (pp. 1864–1871).

Mosterman, P. J., & Zander, J. (2016). Industry 4.0 as a cyber-physical system study. Software & Systems Modeling, 15(1), 17–29.

Mushtaq, H., Al-Ars, Z., & Bertels, K. (2013, December). Accurate and efficient identification of worst-case execution time for multicore processors: A survey. In 8th IEEE design and test symposium (pp. 1–6).

Nicolescu, G., & Mosterman, P. J. (2009). Model-based design for embedded systems. CRC Press.

Nowotsch, J., Paulitsch, M., Bühler, D., Theiling, H., Wegener, S., & Schmidt, M. (2014, July). Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In 26th euromicro conference on real-time systems (pp. 109–118).

Panel, S. G. I. (2010). Guidelines for smart grid cyber security (techreport No. 7628). Cyber Security Working Group (NIST).

Paolino, M., Fanguede, J., Nikolaev, N., & Raho, D. (2016). Turning an open source project into a carrier grade vSwitch, for NFV: VOSYSwitch challenges & results. In 5th IEEE international conference on network infrastructure and digital content (NIDC).

Pellizzoni, R., Meredith, P., Nam, M.-Y., Sun, M., Caccamo, M., & Sha, L. (2009). Handling mixed-criticality in SoC-based real-time embedded systems. In Proceedings of the 7th ACM international conference on embedded software (pp. 235–244). New York, NY, USA: ACM.

Pérez, H., & Gutiérrez, J. J. (2014, March). A survey on standards for real-time distribution middleware. ACM Computing Surveys, 46(4), 49:1–49:39.

Potkonjak, M., Meguerdichian, S., & Wong, J. L. (2010, November). Trusted sensors and remote sensing. In IEEE Sensors (pp. 1104–1107).

Prehofer, C., Horst, O., Dodi, R., Geven, A., Kornaros, G., Montanari, E., & Paolino, M. (2016). Towards trusted apps platforms for open CPS. In 3rd international workshop on emerging ideas and trends in engineering of cyber-physical systems (eitec) (pp. 23–28).

Prehofer, C., Kornaros, G., & Paolino, M. (2015). TAPPS - trusted apps for open cyber-physical systems. In S. K. Katsikas & A. B. Sideridis (Eds.), E-democracy – citizen rights in the world of the new computing

paradigms: 6th international conference, e-democracy 2015, athens, greece, december 10-11, 2015, proceedings (pp. 213–216). Cham: Springer International Publishing.

Richter, A., Herber, C., Rauchfuss, H., Wild, T., & Herkersdorf, A. (2014, February). Performance isolation exposure in virtualized platforms with PCI passthrough I/O sharing. In E. Maehle, K. Römer, W. Karl, & E. Tovar (Eds.), Architecture of computing systems – ARCS 2014: 27th international conference, Proceedings (pp. 171–182). Lübeck, Germany: Springer International Publishing.

Rizzo, L., & Lettieri, G. (2012). Vale, a switched Ethernet for virtual machines. In Proceedings of the 8th international conference on emerging networking experiments and technologies (pp. 61–72).

Ruhland, A., Prehofer, C., & Horst, O. (2016, December). embSFI: An approach for software fault isolation in embedded systems. In M. Völp, P. Esteves-Verissimo, A. Casimiro, & R. Pellizzoni (Eds.), 1st workshop on security and dependability of critical embedded real-time systems (pp. 6–11). Porto, Portugal. Retrieved from https://certs2016.uni.lu/Media/certs2016.uni.lu/Files/CERTS-2016-Ruhland-embSFI (co-located with the IEEE Real-Time Systems Symposium 2016)

Russell, R. (2008). virtio: towards a de-facto standard for virtual I/O devices. ACM SIGOPS Operating Systems Review, 42(5), 95–103.

Sandberg, H., Amin, S., & Johansson, K. H. (2015, February). Cyberphysical security in networked control systems: An introduction to the issue. IEEE Control Systems, 35(1), 20–23.

Sha, L. (2009, September). Resilient mixed-criticality systems. CrossTalk, 22(9-10), 9–14.

Stankovic, J. A. (2014, February). Research directions for the internet of things. IEEE Internet of Things Journal, 1(1), 3–9.

Sun, Y., Han, Z., & Liu, K. J. R. (2008, February). Defense of trust management vulnerabilities in distributed networks. IEEE Communications Magazine, 46(2), 112–119.

Wan, J., Zhang, D., Zhao, S., Yang, L. T., & Lloret, J. (2014, August). Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. IEEE Communications Magazine, 52(8), 106-113.

Wang, Q., Khurana, H., Huang, Y., & Nahrstedt, K. (2009, April). Time valid one-time signature for time-critical multicast data authentication. In IEEE INFOCOM (pp. 1233–1241).

Weimerskirch, A. (2014). V2V communication security: A privacy preserving design for 300 million vehicles. In Proceedings of the workshop on cryptographic hardware and embedded systems (CHES). Busan, Korea.

Zhang, L., & Zhang, H. (2016). A survey on security and privacy in emerging sensor networks: From viewpoint of close-loop. Sensors, 16(4), 443.

Zimmer, C., Bhat, B., Mueller, F., & Mohan, S. (2015). Intrusion detection for CPS real-time controllers. In S. K. Khaitan, J. D. McCalley, & C. C. Liu (Eds.), Cyber physical systems approach to smart electric power grid (pp. 329–358). Berlin, Heidelberg: Springer.