embeddedworld2017
Exhibition&Conference
... it's a smarter world

Security by Design: Introduction to MILS

Sergey Tverdyshev
Research & Development
SYSGO AG
Germany

Abstract **A "security by design" method achieves robustness against programming errors and malicious attacks. A security by design method must be simple to understand. It must be simple to implement, and also to simple to verify. It must enable the developer to create assurance evidence coherent with the design decisions. MILS is a security by design method. In short, application of the MILS approach starts with partitioning the system under design into isolated compartments. System resources, e.g. CPUs, CPU time, memory, IO devices, files, are assigned to compartments. After that the communication channels between compartments are defined with respect to the required API (e.g. POSIX, ARINC, AUTOSAR). Communication and resource sharing between security domains have to be explicit, i.e. everything is forbidden what is not explicitly allowed. In parallel threat modeling is executed, i.e. define system assets to be protected, threat agents and possible malicious actions, system objectives to fight the threats. MILS provides a way to execute mixed-critical applications of different pedigrees on one system. The system as a whole still can be certified to the highest security and safety assurance levels. This makes the approach extremely interesting for modern complex systems, e.g. in a car infotainment system: Android applications can run on the same platform as AUTOSAR applications that communicate with the engine. Until ca. 2000 the MILS concept was mainly used in the US military. Now the commercial interest has picked up. We explain a MILS Architectural Template that simplifies to set up MILS systems. We finish with applications of the MILS concepts across automotive and avionics.**

I. INTRODUCTION

A "security by design" must enable the system developer to create assurance evidence coherently with the design decisions. Thus, when the requirements state "what the system does", the design supports to explain "why the system does it correctly". Moreover, design is largely about splitting a system into components and connections, and security by design shall be able to securely separate components from each other. That is, in case a low-criticality component functionality yields to an attacker, the design shall guarantee that other high-criticality components remain unharmed.

MILS is a high-assurance security architectural approach based on the concepts of separation and controlled information flow. The MILS approach has become a new strategy for effectively designing systems requiring high-assurance, including effective and efficient compositional certification.

II. MILS PLATFORM

The cornerstone of the architecture is a *MILS platform*, which encapsulates trusted and untrusted applications into compartments that reduce mutual dependencies to communications over channels explicitly defined by security policies. This cornerstone component has to be non-bypassable, evaluable, always invoked, and tamperproof (NEAT) [1], [2]. The MILS platform is implemented as a software-hardware (SW-HW) system, consisting of a software separation kernel and the hardware central processing unit (CPU) and memory management unit (MMU), as well as other critical hardware devices (e.g. IOMMU) and their software [3]. The central part of the MILS platform is its *separation kernel*, which implements those separated compartments, manages hardware, as well as enforces security policies for information flow, access control, and resource availability. The MILS principles are based on system decomposition of HW and SW into meaningful and from a security point of view atomic components.

In a benign environment, built for cooperating actors, it suffices if resource decompositions are partial [4]. In an adversarial environment, the allocation of resources has to be completely compartimentalised (see Table 1), such that every resource belongs to some component and can only be accessed by any another component if allowed by the global configuration as well as by the resource owning component itself ("security domains" [5]). Thus, if a component fails or starts acting maliciously, other components are unaffected.

Decom-position means	Components the system is decomposed to	Typical enforce-ment mechanisms	Security view on environment	Security properties preserved	Safety protection against
Functions	Independent/ loosely connected computations	Compiler convention for register storing at function entry / exit; stack convention	Cooperative	None	Uninten-ded reuse of variables
Threads	Single-thread computation	Saving/restoring the threads' states	Cooperative	None	Uninten-ded reuse of variables
Operating system: Processes	Applications / processes in an operating system	Address space separation by MMU	Cooperative	Integrity, confidentiality for memory accesses	Uninten-ded reuse of memory locations
Hypervi-sor: Virtu-alization	Guest operating systems	CPU virtualization	Cooperative to malicious	Integrity, confidentiality for CPU operations	Uninten-ded access to resources
MILS	Partitions	CPU virtualization, real-time, resource management, secure IO, use of a small code base	Malicious	Integrity, confidentia-lity, availability, information flow	Malicious resource depletion

Table 1: Overview of software decomposition means

III. BUILDING MILS SYSTEMS

Architecture is the planning of how to build a new or reengineer an existing system from components [6, 7]. Thus, the first step is the selection of the components (architectural decomposition) of the system into components and connections denoting information flows. Where it is possible to identify components that have no connection, decomposition produces isolated components. Often, components do have one or more connections, and a common convention is to denote components as shapes (e.g., circles) and connections by arrows, as in Figure 1. The absence of an arrow (separation) is as important as its presence. The low-criticality (green) partitions are connected to an untrusted network and high-criticality partitions (red) control an actuator. The separation kernel controls and separates all resources and enforces security policies. It has a small codebase and thus is amenable to high-assurance verification and certification. In summary, MILS gives an agreement to define clear layers, applicable to a large class of systems, although these systems do not all brand themselves MILS.

When applying a MILS design, the integrator of the embedded system using the MILS separation kernel assigns system resources, e.g. CPUs, CPU time, memory, IO devices, files, to compartments, called partitions, and puts applications (that, for instance, may have been developed at different criticality) into the partitions. The integrator may choose to equip certain partitions with additional libraries or run-time environments (e.g. POSIX, ARINC, AUTOSAR) or even to run (para-) virtualized operating systems (e.g. Linux) in a partition. After that, the integrator defines communication channels between partitions with respect to the required API (e.g. POSIX, ARINC, AUTOSAR). Any communication and resource sharing between security domains has to be explicit, i.e. everything is forbidden what is not explicitly allowed.

Thus, the MILS approach provides a way to execute mixed-critical applications of different pedigrees on one system and still have that system being certified to the highest security and safety assurance levels. This makes the approach extremely interesting for modern complex systems, e.g. in a car infotainment system: Android applications can run on the same platform as AUTOSAR applications communicating with the engine.

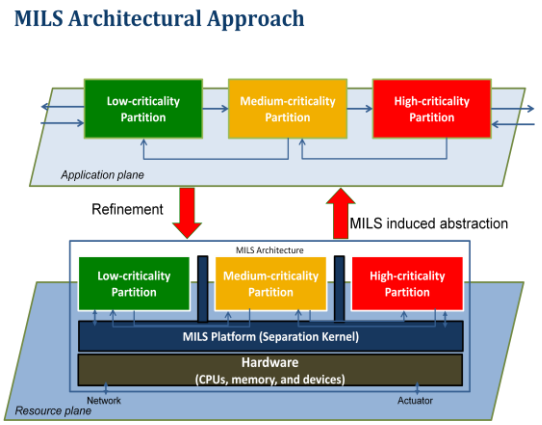

Figure 1: MILS Architectural Approach

The MILS approach enables different composition strategies, comprising:

- T-composition (Figure 2): the composition of the MILS platform with the applications running on the MILS platform.
- P-composition or "puzzle composition" (Figure 3) takes into account maintenance aspects: at time point X, the blue partition has been updated, and the redesign / recertification effort is kept local to the blue partition.

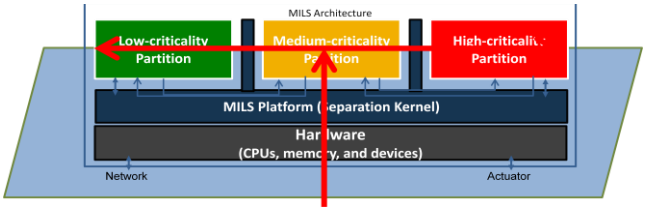

Figure 2: T-composition

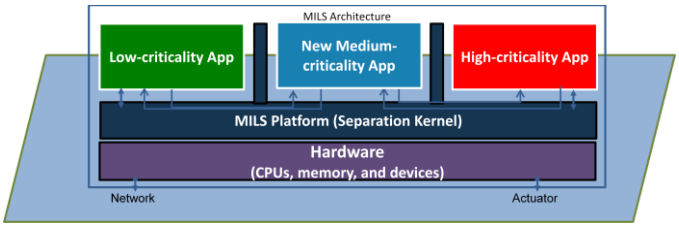

Figure 3: P-composition or puzzle composition

An advantage you gain by using the MILS approach is that it becomes easy to describe the high-level architecture of your system, which can be aligned to the MILS architecture. As first step to standardize the terminology of describing MILS

architecture, a MILS architectural template is available (Figure 4). Figure 4 shows that assurances made by the MILS architecture not only depend on the separation kernel, but also strongly depend on the underlying hardware, a topic that today will be treated for ARM [8] and NXP [9] hardware.

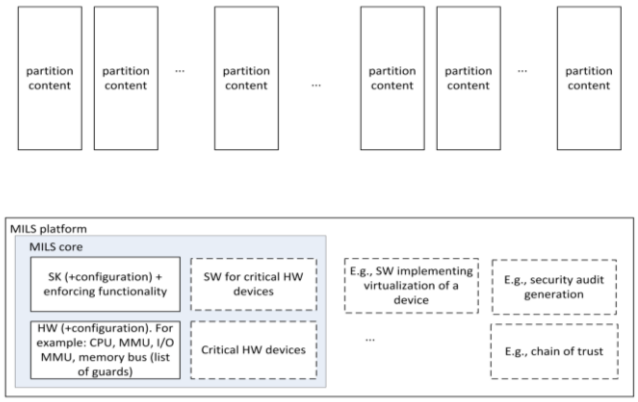

Figure 4: MILS Architectural Template [3]

For instance, in the automotive architecture depicted in Figure 5, on the right-hand side there is a MILS system with PikeOS as separation kernel, a network manager, Android (interacting with a modem), and AUTOSAR (interacting with CAN), implemented as three partitions. Further MILS designs that will be presented today are another automotive design [10], an avionics design [11], a design for critical industrial systems [12], and designs for fog computing [13].

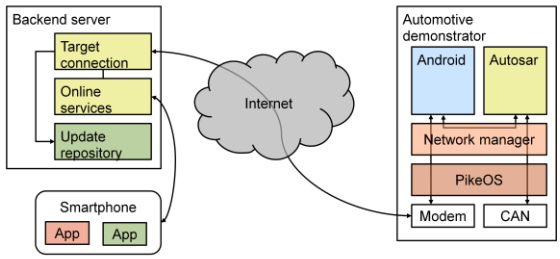

Figure 5: Automotive architecture [14]

IV. Assurance and Certification

The modularity of MILS design simplifies establishing a security / safety case and compositional certification. For instance, for security a draft MILS protection profile has already been established (Figure 6). A compositional safety case is discussed in [15]. MILS design analysis and certification for safety and security will be subject of talks this afternoon [16] [17] [18].

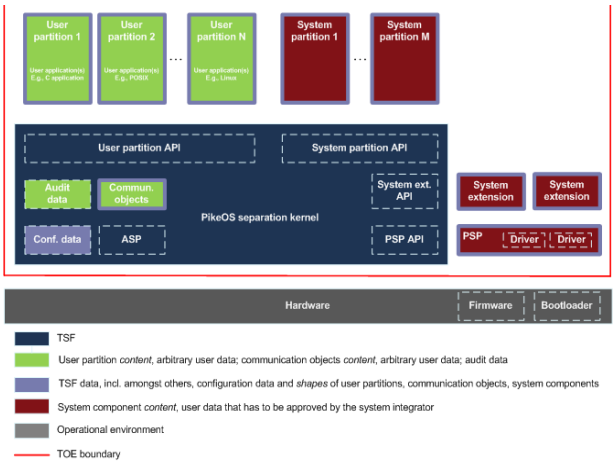

Figure 6: MILS PP [19]

V. Discussion

To support MILS users, a MILS Community has been founded, as an informal group that specifically targets users of MILS systems [20]. For instance, the MILS Community has produced a MILS roadmap (Figure 7). We expect today's workshop to push forward common understanding of MILS topics, including building systems with the MILS architecture, as well as bringing forward new approaches for MILS foundations and MILS verification.

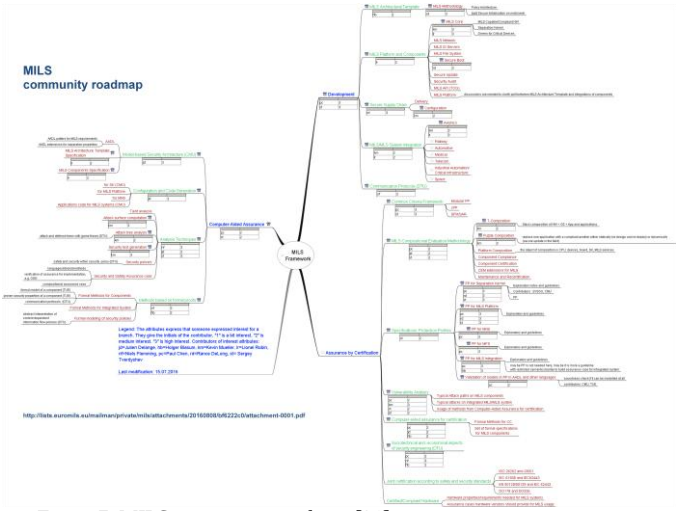

Figure 7: MILS community roadmap [21]

VI. Acknowledgment

Work has received partial funding from the EU Horizon 2020 projects CITADEL, http://www.citadel-project.org/ (project number 700665) and certMILS http://www.certmils.eu/, (project number 731456).

VII. Bibliography

[1] D. Kleidermacher and M. Wolf, "MILS Virtualization for Integrated Modular Avionics," in *Digital Avionics Systems Conference (DASC)*, St Paul, 2008.

[2] J. Rushby, "Design and Verification of Secure Systems," 1981. [Online]. Available: http://www.sdl.sri.com/papers/sosp81/sosp81.pdf.

[3] S. Tverdyshev, H. Blasum, B. Langenstein, J. Maebe, B. De Sutter, B. Leconte, B. Triquet, K. Mueller, M. Paulitsch, A. Soeding-Freiherr von Blomberg and A. Tillequin, “MILS Architecture,” 2013. [Online]. Available: http://dx.doi.org/10.5281/zenodo.45164.

[4] H. A. Simon, “The Architecture of Complexity,” *Proc Am Phil Society*, vol. 106, no. 6, pp. 467-482, December 1962.

[5] B. W. Lampson, “Protection,” in *Proc Fifth Annual Princeton Conference on Information Sciences and Systems*, Princeton, 1971.

[6] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003.

[7] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them," *Software Engineering, IEEE Transactions on*, vol. 21, no. 4, pp. 314-335, 1995.

[8] M. Meriac, “High-End Security Features for Low-End Microcontrollers: Hardware-Security Acceleration on ARMv8-M Systems,” this workshop, 2017.

[9] G. Waters, “Hardware Enforced Separation in Embedded Multicore SoCs,” this workshop, 2017.

[10] E. Waitz, “Current Trends and Solutions in Securing Automotive Software,” this workshop., 2017.

[11] K. Müller, “Hardening High Assurance Systems: MILS as Software Design for Avionics,” this workshop, 2017.

[12] E. Rudina, “An Approach to SoD Validation for MILS Security Configurations,” this workshop, 2017.

[13] W. Steiner, “Fog Computing as Enabler for the Industrial Internet of Things /,” this workshop, 2017.

[14] S. Tverdyshev, “EURO-MILS: Building and certifying modular secure systems,” 2016. [Online]. Available: https://doi.org/10.5281/zenodo.47972.

[15] K. Netkachova, K. Müller, M. Paulitsch and R. Bloomfield, “Security-Informed Safety Case Approach to Analysing MILS Systems,” 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47987.

[16] I. Furgel, V. Saftig, T. Wagner, K. Müller, R. Schwarz and A. Söding-Freiherr von Blomberg, “Non-Interfering Composed Evaluation,” this workshop, 2017.

[17] S. Nordhoff and H. Blasum, “Ease Standard Compliance by Technical Means via MILS,” this workshop, 2017.

[18] T. Noll, “Analysing Cryptographically-Masked Information Flows in MILS-AADL,” this workshop, 2017.

[19] I. Furgel and V. Saftig, “D12.3 Common Criteria Protection Profile, “Multiple Independent Levels of Security: Operating System” (MILS PP: Operating System),” 2016. [Online]. Available: https://doi.org/10.5281/zenodo.51582.

[20] MILS Community, “MILS Community web site,” [Online]. Available: http://mils-community.euromils.eu/.

[21] MILS Community, “MILS Community Roadmap,” 2016. [Online]. Available: http://lists.euromils.eu/mailman/private/mils/attachments/20160808/bf6222c0/attachment-0001.pdf .