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

Ease Standard Compliance by Technical Means via MILS

Sven Nordhoff
Director Certification
SYSGO AG
Germany

Holger Blasum
Research & Development
SYSGO AG
Germany

***Abstract*—You have to develop an embedded system? You need to show its conformance to a safety standard (e.g. IEC 61508, ISO 26262, DO-178) or a security standard (e.g. IEC 62443, Common Criteria)? How does your life get easier by using a MILS design? Using an embedded operating system can help with modularization. Moreover, a *MILS* embedded operating system isolates processes and their resources from each other. Resource management and information flow control enable separation in time and separation in space. In this paper we show standard compliance work units that MILS helps achieving by technical means.**

Keywords—MILS (Multiple Independent Levels of Security/Safety) operating system, system using MILS, separation kernel, safety, security, embedded system, IEC 61508, ISO 26262, DO-178, IEC 62443, Common Criteria

I. INTRODUCTION

Our objective is practical: assume you develop an embedded system and want to show its conformance to a safety / security standard (e.g. IEC 61508, ISO 26262, DO-178 IEC 62443, or Common Criteria), how does your life get easier by using a MILS design?

At first glance, in the above-mentioned standards, there is no hard requirement like "you shall use a MILS design". However, software developer common sense is that it is a good idea to develop a system in a modular way. The same common sense will be to reuse existing (COTS = common off-the-shelf) components. In particular, it is a good idea to use an operating system to abstract hardware. Unsurprisingly, this common sense is reflected in the standards themselves. We will point to those sections within the standards where this reasoning can be found. Up to this point, the arguments are valid for using *any* COTS operating system.

A MILS operating system (also called "separation kernel" [1]) provides controlled resource management and information flow, which are used for separation in time and separation in space. Using a MILS operating system allows to isolate processes and their resources from each other, allowing to show that processes do not interfere with each other or their resources. This means that the burden of proof for separation properties is shifted away from the developer of a product using a MILS operating system. Viewed from the perspective of verification, a MILS operating system reduces the number of possible states your embedded system may have. The adequate qualification and verification of the MILS operating system has been taken care of by the MILS operating system vendor.

We point to where and to which extent separation in time and space can be found in the different safety and security standards and how, in our eyes, the use of a MILS operating system simplifies to get credit for the fulfillment of certain requirements of standards, which, in some cases, would be much harder to confidently fulfill without a MILS operating system.

II. RELEVANT STANDARDS

In this paper we have analyzed standards that are very common and in most cases official means of compliance in different industries. All these standards strongly benefit from using a MILS architecture:

IEC 61508 [2] is applicable for safety certification of a broad range of cyber-physical systems, it covers software and hardware. This standard is used in all industrial domains.

ISO 26262 [3] is an instantiation of IEC 61508, specifically targeting the automotive sector. It covers software and hardware.

DO-178 [4] is "the" standard for qualification of airborne software. It mainly covers software, but also emphasizes the relation of the software to the system (including hardware).

IEC 62443 [5] is directly targeting security for IEC 61508. IEC 61508 Part 1, Section 7.5.2.2, states that "if security

threats have been identified, then a vulnerability analysis should be undertaken in order to specify security requirements" and points to IEC 62443. As of January 2017, some parts of IEC 62443 still are under development (for which working drafts are available). It covers software and hardware.

Lastly, the Common Criteria for Information Technology Security Evaluation [6] (abbreviated as "CC") is a standard for security, targeting software and hardware.

III. MILS CONCEPTS AND TECHNICAL MEANS IN THE STANDARDS

MILS is an architecture to build highly reliable systems with a high degree of assurance. MILS achieves modularity and well-defined flows by using partitioning, access control, resource management and control of information flow [7].

A. Modularity and well-defined flows

All standards mentioned above emphasize that a simple, understandable, and verifiable design is helpful to achieve the desired security and safety properties.

One of the most widely accepted techniques to ensure a simple, understandable, and verifiable design is to partition the design into different subsystems. This concept occurs in all standards. Of course, it is usually not possible nor desirable to break down a design into completely isolated components; rather components probably will interact via well-defined control and data flow. Modularity and flow control is one of the tasks that MILS operating systems support well.

B. Space partitioning, time partitioning

Modularity can only be achieved when resources are allocated clearly and non-bypassably. A MILS operating system is a *partitioned* embedded operating system, and this is one point where MILS operating systems are stronger than other embedded operating systems. Space partitioning is the static allocation of resources (such as memory). Time partitioning comprises handling of interrupts, and scheduling, so that time quotas are always ensured. IEC 61508 explicitly distinguishes between separation in space and separation in time. If separation in space is assumed to comprise "memory access control" and separation in time to comprise "cyclic scheduling", then a similar distinction, albeit with different wording, is found in DO-178, ISO 26262, IEC 62443 and the CC. Table 1 gives examples where to find space partitioning and time partitioning in the standards.

Standard	Space	Time
IEC 61508, Part 3, Annex F (F2, F4, F5)	"Spatial: the data used by one element shall not be changed by another element. In particular, it shall not be changed by a non-safety related element. a) Use of hardware memory protection between different elements, including elements of differing systematic capability. b) Use of an operating system which permits each element to execute in its own process with its own virtual memory space, supported by hardware memory protection. c) Use of rigorous design, source code and possibly object code analysis to demonstrate that no explicit or implicit memory references are made from between software elements which can result in data belonging to another element being overwritten (for the case where hardware memory protection is not available). d) Software protection of the data of a higher integrity element from illegal modification by a lower integrity element."	"Temporal: one element shall not cause another element to function incorrectly by taking too high a share of the available processor execution time, or by blocking execution of the other element by locking a shared resource of some kind. a) Deterministic scheduling methods. For example, a cyclic scheduling algorithm which gives each element a defined time slice supported by worst case execution time analysis of each element to demonstrate statically that the timing requirements for each element are met; time triggered architectures. b) Strict priority based scheduling implemented by a real-time executive with a means of avoiding priority inversion. c) Time fences which will terminate the execution of an element if it over-runs its allotted execution time or deadline (in such a case, hazard analysis shall be undertaken to show that termination of an element will not result in a dangerous failure, so this technique may be best employed for a non-safety related element). d) An operating system which guarantees that no process can be starved of processor time, for example by means of time slicing. Such an approach may only be applicable where there are no hard real time requirements to be met by the safety related elements, and it is shown that the scheduling algorithm will not result in undue delays to any element."
ISO 26262, Part 6, Annex D	"Memory: With respect to memory, the effects of faults such as those listed below can be considered for software elements executed in each software partition: • corruption of content; • read or write access to memory allocated to another software element. EXAMPLE Mechanisms such as memory protection, parity bits, error-correcting code (ECC), cyclic redundancy check (CRC), redundant storage, restricted access to memory, static analysis of memory accessing software and static allocation can be used." (Note: SIL D needs hardware support, see Part 6, 7.4.11.)	"Timing and execution: With respect to timing constraints, the effects of faults such as those listed below can be considered for the software elements executed in each software partition: • blocking of execution; • deadlocks; • livelocks; • incorrect allocation of execution time; • incorrect synchronization between software elements. EXAMPLE Mechanisms such as cyclic execution scheduling, fixed priority based scheduling, time triggered scheduling, monitoring of processor execution time, program sequence monitoring and arrival rate monitoring can be considered."
DO-178, Section 2.4.1	"A partitioned software component should not be allowed to contaminate another partitioned software component's code, input/output (I/O), or data storage areas." (Note: A more explicit distinction between space and time can be found in ARINC-653 [8].)	"A partitioned software component should be allowed to consume shared processor resources only during its scheduled period of execution."
IEC 62443, Part 3.3	Space partitioning not explicitly mentioned, but implied by partitioning functional requirements.	Time partitioning not explicitly mentioned, but implied by resource management functional requirements.
CC, Part 2	Space partitioning not explicitly mentioned, but implied by access control (FDP_ACC, FDP_ACF).	Time partitioning not explicitly mentioned, but implied by resource management (FRU_RSA).

Table 1: Space and time partitioning in different standards

C. *Partitioning, freedom from interference, non-interference, information flow*

DO-178 Section 2.4.1 explicitly describes partitioning and IEC 61508 Part 3, Annex F, describes a similar concept: "independence of execution" and also "non-interference". ISO 26262 Part 6, Annex D, uses the term "freedom from interference". In DO-178 and IEC 61508, non-interference is geared towards availability and integrity. That is, the integrity of each application and the availability of the resources it uses are not hindered by other applications. ISO 26262 Part 6, Annex D also stresses integrity and availability, but additionally includes to freedom from interference the absence of accesses to memory allocated to another software element (i.e. not only covering write access, but also read access). No such explicit descriptions have been found in IEC 62443, but related functionality in 62443 is application (or device) partitioning, denial of service protection and resource management (SR 5.4, CR 5.4, SR 7.1, CR 7.1, SR 7.2, CR 7.2). The CC Part 2 provides support to cover functionality giving non-interference by access control (FDP_ACC, FDP_ACF) information flow (FDP_IFC, FDP_IFF), and/or resource management (FRU_RSA).

IV. STANDARD COMPLIANCE WORK UNITS THAT MILS HELPS ACHIEVING BY TECHNICAL MEANS

In this section, we assume that a system will be built that uses a MILS operating system, and the software is encapsulated into different, possibly communicating partitions. For an avionic example of such a system see e.g. [9], for an automotive example see e.g. [10]. The focus of the analysis in this section is on requirements, architecture and design, it is not on verification/testing.

A. *ISO 61508 [2]*

Within ISO 61508 our focus is Part 3, which deals with software. We begin with Part 3 Section 7.2.2 that is on software safety requirements specification. Here MILS particularly eases the specification of "any safety-related or relevant constraints between the hardware and the software" (7.2.2.7), to "clearly identify the non-safety functions" (7.2.2.9), identify "functions related to the detection, annunciation and management of faults in the software itself (software self-monitoring)" (7.2.2.10, by use of MILS health monitoring provided by the MILS operating system), to fulfill "independence requirements between functions" (7.2.2.10), and to analyze "best case and worst case execution time" (7.2.2.12, the analysis is simplified by using time partitioning). Part 3 Section 7.4.3 (requirements for software architecture design) gives "operating systems" as example of "major software elements" that can be element of a software architecture design. Part 3 section 7.4.3.2.b states that a software architecture shall "be based on a partitioning into elements/subsystems", moreover a focus of software architecture (7.4.3.2.c) is to "determine all software/hardware interactions and evaluate and detail their significance". The use of a MILS operating system by design gives a technical separation into partitions and all software/hardware interactions can be traced to the level of partitions.

Part 3 Section 7.4.2 is dealing with software design requirements, here MILS is particularly useful for "abstraction, modularity and other features which control complexity", "the expression of ... information flow between elements, ... timing constraints" (7.4.2.2). MILS is a feature that "facilitates software modification" (7.4.2.4, by puzzle composition [11]), and it allows to "keep the safety-related part of the software simple" (7.4.2.6, by factoring out the safety-related part of software into a high-criticality partition), and provides "adequate design measures ensure that the failures of non-safety functions cannot adversely affect safety functions" (7.4.2.8), "unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design" (7.4.2.9, use a MILS platform to justify independence).

B. *ISO 26262 [3]*

Here it is Part 6 that focuses on software. For the specification one has to show "compliance and consistency with the technical safety requirements; compliance with the system design; and consistency with the hardware-software interface" (Section 6.4.8) which, again is much easier when the system design consists of independent partitions.

Part 6 Section 7.4 is on software architecture. It is required to take into account "the testability of the software architecture during software integration testing; and the maintainability of the software architectural design" (Section 7.4.2), which are simplified by MILS design (thanks to puzzle composition), the design shall exhibit "modularity; encapsulation; and simplicity" (Section 7.4.3), the software architectural design shall describe "the functionality and behavior; the control flow and concurrency of processes; the data flow between the software components; the data flow at external interfaces; and the temporal constraints" (Section 7.4.5) which again is much easier if you have a partitioned system to begin with. Section 7.4.11 explicitly deals with software partitioning, requiring (amongst other) "that shared resources are used in such a way that freedom from interference of software partitions is ensured" (provided by MILS resource separation). Lastly, MILS time and space partitioning greatly helps to ensure the requirements of 7.4.17: "An upper estimation of required resources for the embedded software shall be made, including: the execution time; the storage space; and the communication resources."

C. *DO-178 [4]*

Section 2.3 states that "only partitioned software components can be assigned individual software levels by the system safety assessment process." Section 2.4.1 clarifies that "a partitioned software component should not be allowed to contaminate another partitioned software component's code, input/output (I/O), or data storage areas", that it "should be allowed to consume shared processor resources only during its scheduled period of execution" and "failures of hardware unique to a partitioned software component should not cause adverse effects on other partitioned software components",

which are a strong separation properties again provided by MILS systems.

On software architecture DO-178 Section 6.3.3.a (referenced from Table A-4, item 8) states that functions that provide (logical) partitioning do not conflict with high-level requirements, one way of doing this is to use a MILS operating system. Section 6.3.3.b (referenced from Table A-4, item 9) describes that "the objective is to ensure that a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow." MILS provides a good control of data flow and control flow. Further objectives comprise "initialization, asynchronous operation, synchronization, and interrupts" (Section 6.3.3.c, referenced from Table A-4, item 10, which a MILS system can well control), "software architecture is verifiable" (Section 6.3.3.d, referenced from Table A-4, item 11, MILS system break down complexity to partitions and ease verifiability) and that "software partitioning integrity is confirmed" (Section 6.3.3.f, referenced from Table A-4, item 13, MILS systems allow to ensure adequate partitioning integrity by choice of a MILS operating systems).

D. IEC 62443 [5]

As mentioned before, IEC 62443 is not yet fully finalized. Our discussion mainly builds on Part 4.1 (Draft 3, Edit 10) for the requirements for secure product development lifecycle requirements.

Part 4.1 Section 7.3 SR-2 and higher demands that "all products shall have an up-to-date threat model with the following characteristics: correct flow of categorized information throughout the system," including "trust boundaries". Use of a MILS system provides partitions as natural application containers with "trust boundaries".

Part 4.1 Section 8.3 SD-2 (defense in depth in design enhancement level 2) demands that "a process shall be employed for including multiple layers of defense where each layer provides additional defense mechanisms. Each layer should assume that the layer in front of it may be compromised. Secure design principles are applied to each layer." A MILS operating system naturally lends itself be used to add one layer of a defense. For SD-6 (Part 4.1 Section 8.7), MILS covers "least privilege", "using proven secure components/designs where possible", "economy of mechanism (striving for simple designs),", "using secure design patterns", "attack surface reduction", "all trust boundaries are documented as part of the design".

In terms of functional requirements (Part 3.3 and Part 4.2 of IEC 62443), MILS very well addresses application (or device) partitioning (Part 3.3 SR 5.4 / Part 4.2 CR 5.4), denial of service protection (Part 3.3 SR 7.1 / Part 4.2 CR 7.1) and resource management (Part 3.3 SR 7.2 / Part 4.2 CR 7.2).

E. Common Criteria (CC) [6]

In terms of product security functional requirements (CC Part 2), MILS systems make it easy to show access control (FDP_ACC, FPD_ACF), information flow control (FDP_IFC, FPD_IFF), and resource management (FRU_RSA), for an example see [12].

For methodology, we base our discussion on CEM (Common Evaluation Methodology), which directly addresses CC evaluators. For the product design, ADV_TDS.3-1 demands that "the structure of the entire" product "is described in terms of subsystems" and ADV_TDS.3-6 demands that the product documentation shall ensure that "interactions between the subsystems" of the product are described. In a MILS-based system, partitions are natural candidates for subsystems.

For the product architecture, ADV_ARC.1-2 demands that the evaluator looks at the "security domains". If you use a partitioned system, then partitions are building blocks for security domains (see e.g. [9]). ADV_ARC.1-4 demands to check how the product protects "itself from tampering by untrusted active entities" and ADV_ARC.1-5 demands that the security architecture adequately describes how the security-enforcing functionality "cannot be bypassed". Here again, it comes handy to delegate the responsibility to an underlying MILS operating system.

V. DISCUSSION

We have shown, for each standard, where we see significant potential by using a MILS design. In effect, the standard sections we quoted are a list of assurance activities to be achieved where the burden of proof of tricky architectural properties such as separation, information flow control can be shifted in a concrete embedded system deployment to a MILS operating system. As a benefit, the MILS operating system vendor can provide this assurance for the MILS operating system as a product. Of course, we do not claim that the above list is complete, e.g. testing has been largely omitted from the analysis. That omission is mainly because we feel that the benefits for testing are more indirectly resulting from the simplified architecture rather than direct requirements in the standards themselves that a partitioned architecture needs to be used for testing.

For each standard, here, so far, for simplicity we have focused on the software design itself. Often, other parts of the standards than the above-mentioned software-centric parts deal with software-hardware interactions and more arguments for the use of MILS systems can be found. E.g. the automotive ISO 26262 Part 4 ("Product development at the system level") Section 7.4.6 states that for hardware-software interaction one has to specify "the hardware features that ensure the independence between elements and that support software partitioning; shared and exclusive use of hardware resources; the access mechanism to hardware devices; and the timing constraints defined for each service involved in the technical safety concept", all which is made much simpler when the hardware interaction can be confined either to the MILS operating system or single driver partitions.

All standards require the common sense that if partitioning is used for separation of critical from non-critical components, then the software implementing the partitioning shall be as trustworthy (in terms of assurance level) as is the most critical component that is being partitioned. That is if you have, an application with criticality X, then the MILS operating system at least shall have assurance level X.

Compositional certification is the reuse of artifacts for a lower-level component for a higher-level component (e.g. a system based on a MILS operating system reuses assurance of the MILS operating system), and it is established practice e.g. for DO-178, where MILS systems are frequently deployed. However, each DO-178 certification only targets a specific deployment, and, for software, is usually done fully white-box. The CC as well as IEC 61508, ISO 26262, and IEC 62443 are product-centric, thus they offer reuse and compositional certification (see e.g. [13], [14]).

VI. Acknowledgment

Work by one of the authors has received partial funding from EU Horizon 2020 project certMILS http://www.certmils.eu/, project number: 731456.

VII. Bibliography

[1] S. Tverdyshev, "Security by Design: Introduction to MILS," in *this workshop*, 2017.

[2] International Electrotechnical Commission, Technical Committee 65: Industrial-process measurement and control, IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2.0 ed., 3, rue de Varembé, CH-1211 Geneva 20: IEC Central Office, 2010.

[3] ISO/TC 22, Road vehicles, Subcommittee SC 3, Electrical and electronic equipment, ISO 26262 Road vehicles - Functional safety, Case Postale 56, Geneva, Switzerland: International Standards Organization, 2011.

[4] RTCA SC-205 / EUROCAE WG-71, DO-178C: Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics (RTCA), Inc., 1150 18th NW, Suite 910, Washington, D.C. 20036, 2011.

[5] International Electrotechnical Commission, Technical Committee 65: Industrial-process measurement and control, IEC 62443: Security for industrial automation and control systems, 3, rue de Varembé, CH-1211 Geneva 20: IEC Central Office, 2008.

[6] Common Criteria Sponsoring Organizations, *Common Criteria for Information Technology Security Evaluation. Version 3.1, revision 4*, 2012.

[7] 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.

[8] Airlines Electronic Engineering Committee (ARINC), Avionics application software standard interface: ARINC specification 653, Annapolis, MD 21401: Aeronautical Radio, Inc., 1997.

[9] K. Müller, M. Paulitsch, S. Tverdyshev and H. Blasum, "MILS-Related Information Flow Control in the Avionic Domain: A View on Security-Enhancing Software Architectures," 2012. [Online].

[10] I. Furgel, V. Saftig, T. Wagner, K. Müller and R. S. von Blomberg, "Non-Interfering Composed Evaluation," Jan 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47979.

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

[12] K. Müller, M. Paulitsch, R. Schwarz, S. Tverdyshev and H. Blasum, "MILS-Based Information Flow Control in the Avionic Domain: A Case Study on Compositional Architecture and Verification," 2012. [Online].

[13] A. D. Sinnhofer, W. Raschke, C. Steger and C. Kreiner, "Evaluation paradigm selection according to Common Criteria for an incremental product development," 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47986.

[14] EURO-MILS, "Euromils deliverable - D33.1: Addendum to CEM," [Online]. Available: https://doi.org/10.5281/zenodo.47298.