Partitioning in Safety and Security: Mapping to MILS Core Partitioning Mechanisms

Holger Blasum
holger.blasum@sysgo.com SYSGO AG
Klein-Winternheim, Germany

ABSTRACT

While safety and security at a high-level are white-board concepts, once it comes to implementation in a MILS core (separation kernel + minimal set of additional hardware and software needed for the separation of partitions), sometimes the concrete realization depends on what is doable. Several use cases of partitioning are mapped to partitioning mechanisms implementing the partitioning. The main idea is that this paper takes "safety" + "something" and calls it "security". The main result is that different use cases of safety and security can be compared, and one can precisely talk about differences. If one is aware of differences, one can strive for and maintain strong notions of safety and security.

1. INTRODUCTION

This paper explains safety and security separation concepts by mapping them to concrete (functional) partitioning mechanisms that can be traced to an implementation.

The logical unit of partitioning is a "partition". A partition is a container that separates its contents from all other partitions. Optionally, a system may maintain additional measures of separation *within* a partition (for example that multiple threads are maintained within a partition). However, no claim is made with to regards what happens *within* a partition in terms of strict separation. For example, when talking about multiple threads it can be *desired* that they are not strictly separated, for example that the threads share the same address space.

The "MILS core" refers to the minimal set of components needed for separation of partitions on a MILS platform. The only goal of the MILS core is to provide separated partitions with controlled information flow between them. Thus, the MILS core provides the primary security functionality of a MILS system. The MILS core (Figure 1) consists of components that implement and enforce the separation both in space and time. [3, Section 3.2.14].

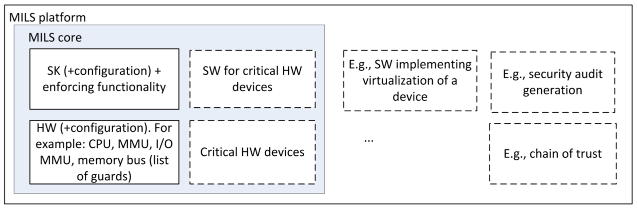
MILS platform
MILS core
SK (+configuration) + enforcing functionality
SW for critical HW devices
HW (+configuration). For example: CPU, MMU, I/O MMU, memory bus (list of guards)
Critical HW devices
E.g., SW implementing virtualization of a device
E.g., security audit generation
...
E.g., chain of trust

Figure 1: MILS core

The ensuing sections discuss the realization of each safety and security use case in terms of partitioning mechanisms. The idea of this paper is that by assigning concrete partitioning mechanisms to separation use cases, one can check what is needed for partitioning in different security scenarios. In other words, this paper takes "safety" + "something" and calls it "security".

From Section 2 to Section 5, the number of partitioning mechanisms is monotonously increasing. We start out with general purpose operating systems (Section 2), and, via real-time safety partitioning kernels (Section 3) go to different use cases of security by MILS cores (Sections 4 and 5). Safety is a good way-point, because for example general-purpose operating systems and the ARINC 653 standard are well-understood. For the sake of the argument, the implicit assumption is that, when they are reduced to the partitioning mechanisms discussed here, MILS cores can be seen as a subset of real-time safety partitioning kernels (+ hardware), and that safety partitioning kernels (+ hardware) can be seen as a subset of general-purpose operating systems.

2. PARTITIONING MECHANISMS FOR SELF-PROTECTION OF GENERAL-PURPOSE OPERATING SYSTEMS

General-purpose operating systems already have some partitioning mechanisms, to protect themselves against user applications.

2.1 Relevant Partitioning Mechanisms

Access control to user space memory (ACU). One resource to partition is the user-space memory (memory that is made available to users for direct access via CPU load/store instructions), this is usually done by address spaces.

Definition: access control to user space memory is the configuration of the MMU and, if applicable, of the IO/MMU, to restrict accesses of applications to the user space memory (addresses the operating system kernel does not reside in). The MMU is part of hardware, the configuration of the MMU is part of the operating system kernel.

Address spaces made available to users (partitions) are typically protected by access control (MMU and I/O MMU). While implemented in hardware, access control is found in all commercially available MMUs, because the effort for the access control (one additional decision more) is small in comparison to the effort for the look-up of a memory cell: for a look-up of one byte in one MB ($= 2^{20}$ bytes) of memory, at least 20 binary decisions have to be taken.

Access control to management data (ACM). To manage the user applications, an operating system kernel maintains certain *management data*. For example, this management data may contain identifiers for applications, and an access control matrix to regulate which objects the applications may access. For resources on which quotas are enforced, the management data contains an upper bound for resource usage, and it may contain data of actual usage: for example, a decision, whether memory can be granted to an application requesting it depends on the assigned quota and how much memory is already used at a given time point. As further example, other management data is used to describe the state of threads that are running in the kernel.

Definition: access control to management data is the configuration of the MMU and, if applicable, of the IO/MMU, to restrict accesses of applications to the kernel memory (address space(s) that the operating system kernel resides in). Again, the MMU is part of hardware, the configuration of the MMU is part of the operating system kernel.

A common design is to store management data in the address space of the operating system. Management data includes the scheduling data, so that a partition cannot influence the schedule.

CPU reuse (CR). In general-purpose operating systems the partitioning mechanism for CPU reuse between different applications ("context switches") stores CPU registers to memory, and loads another set of stored CPU registers from memory, overwriting the previous ones.

Definition: CPU reuse is the reuse of CPU registers, including appropriate initialization and termination, so that residual information is made unavailable on context switch. The mechanism of CPU reuse is usually implemented in the separation kernel. The separation kernel may make use of hardware-specific instructions for context switches if so provided by the underlying hardware (CPU).

Note: we use a strong definition here, because at least for CPU context switches, it is already fulfilled by many safety or general-purpose operating systems.

2.2 Use of Partitioning Mechanisms

Table 1 gives an overview of partitioning mechanisms frequently found in general purpose operating systems.

Resource	Partitioning mechanism(s)
User-space memory	ACU (used for control of writes)
Management data	ACM (used for control of writes)
CPU registers	CR

Table 1: Partitioning mechanisms in general purpose operating systems

3. PARTITIONING FOR REAL-TIME SAFETY

Safety has been defined as the absence of catastrophic failures in a system (e.g. [11]), measured as time to catastrophic failure.

Software safety can be established amongst others (such as development process, verification and validation) by partitioning, characterized in the avionics software development standard DO-178C as technique for providing isolation between software components to contain or isolate faults, potentially reducing the software verification effort. "A partitioned software component should not be allowed to contaminate another component's code, I/O or data storage areas." [15, Section 2.4.1]. Real-time partitioning includes temporal assurance. One of the standards for avionics partitioning systems is ARINC 653 [1]. While the bulk of ARINC 653 focuses on a portable API specification, it [1, Section 2] also summarizes separation requirements, for example it specifies that the OS shall be able to restrict to individual partitions memory spaces, processing time and access to I/O.

Partitioning for real-time safety is implemented by partitioning kernels, which are treated as special case of general-purpose operating systems in this paper.

3.1 Relevant Partitioning Mechanisms

Quotas for memory (QM). In a general purpose operating system such as Linux, by default, the user is considered benevolent with regard memory usage and no further precaution is taken. Thus, it is possible to inadvertently or maliciously bring down a system (for example by a "fork bomb"). Moreover, in addition to user-space memory, to maintain, for example, threads, an operating system needs additional storage space in the kernel, for example to store away registers at a context switch. A partitioning kernel has to make sure that no partition can deplete this storage space, which can be implemented by assigning quotas to the kernel storage space.

Definition: Quotas for memory means that quotas for the amount of user space memory kernel memory used are assigned to each partition. Quotas for memory are implemented by the partitioning kernel. Quotas for memory also can be implemented by completely static assignment of memory.

Cyclical Schedule (CS). In a reactive system, one step to guarantee an upper bound for latency is to establish a fixed cyclical schedule of partitions.

Definition: A cyclical schedule ensures that each partition gets access to CPU(s) on a cyclic basis (infinite repetition of a fixed schedule). A cyclical schedule is implemented by the separation kernel.

Worst-case execution time analysis (WCET). The partitioning mechanism of worst-case execution time (WCET) analysis says that each kernel entry has a worst-case execution time. WCET complements the cyclical schedule: the latency is bounded by $CS + WCET$. WCET can be established by static analysis of source code and hardware testing. As the static analysis is usually done over all source code, usually the effort for static analysis is related to the code size of the system for which static analysis is done: a small system requires less effort, a large system requires more effort. WCET analysis is not translated into run-time instructions.

Definition: worst-case execution time analysis is the property of a system to have been evaluated for an upper bound of worst-case execution time. WCET is done on the whole MILS core and its configuration.

3.2 Use of Partitioning Mechanisms

ARINC 653 explicitly stipulates that user-space memory and user application CPU time are separated; implying also the management of kernel resources, management data and CPU registers.

Table 2 gives an overview of partitioning mechanisms that can be used for ARINC 653 implementations.

Resource	Partitioning mechanism(s)
User-space memory	ACU (used for control of writes)
Management data	ACM (used for control of writes)
CPU registers	CR
Kernel resources	QM
CPU time (latency)	CS + WCET

Table 2: Partitioning mechanisms for partitioning for real-time safety

4. SAFETY + CONFIDENTIALITY BY MMU

Security is usually defined (e.g. [11]) by including some mechanism able to prevent unauthorized disclosure (confidentiality), that could be exploited by a malicious (human) attacker.

DO-178C states that: “A partitioned software component should *not be allowed to contaminate* another component’s code, I/O or data storage areas.” [15, Section 2.4.1] (emphasis ours). The requirement not to contaminate does not imply any requirement not to eavesdrop. Similarly [1, Section 2.3.1] states “Memory partitioning is ensured by prohibiting memory accesses (at a minimum, *write* access) outside of a partition’s defined memory areas.”

However, as access control is in place anyway, it is straightforward to specify that it not only shall forbid “writes” but also “reads”. By this simple measure we enter the realm of security, because we are addressing intelligent observers: it has becoming harder for a malicious attacker to observe the kernel state, and to read out other partitions. Partitioning for real-time safety + some degree of security is implemented by separation kernels, which are treated as special case of partitioning kernels in this paper.

4.1 Additional Relevant Partitioning Mechanisms

The same partitioning mechanisms as for safety are reused (introduced above in Section 3.1). Therefore, for this section, there is *no* additional partitioning mechanism to be introduced.

4.2 Use of Partitioning Mechanisms

The only difference to pure safety is that the access to management data and each partition’s user space memory is configured not only to the effect that other partitions do not *write* to it, but also to the effect that other partitions do to not *read* from it.

An overview of partitioning mechanisms is given by Table 3 below.

Resource	Partitioning mechanism(s)
User-space memory	ACU (used for control of reads and writes)
Management data	ACM (used for control of reads and writes)
CPU registers	CR
Kernel resources	QM
CPU time (latency)	CS + WCET

Table 3: Partitioning mechanisms for safety + confidentiality by MMU

5. CONTROL OF INFORMATION FLOW BETWEEN COLLUDERS

Information flow between colluders that is not authorized is called a covert channel.

A covert channel often depends on slightly altering the system’s behavior. This can even happen to systems that are robust against manipulations of their integrity: an example is given for the Fiasco.OC microkernel in [14, Section IV.C] where it is exploited that in the Fiasco.OC microkernel there exist global read-only pages and, secondly, that any page of memory can be mapped at most 2047 times. An information flow channel is constructed by exploiting that, if partition a_1 has mapped such a read-only page 2047 times, then an attempt to map it from partition a_2 will fail. If a_1 has mapped such a read-only page less than 2047 times, then an attempt to map it from partition a_2 will succeed.

This section targets attack scenarios that assume that at least *two* partitions cooperate ("collude") on attacks.

5.1 Additional Relevant Partitioning Mechanisms

Access control to kernel resources (ACK). With a storage channel the sending partition alters a particular data item, and the receiving partition detects and interprets the value of the altered data to receive information covertly. With a timing channel the sending partition modulates the amount of time required for the receiving process to perform a task or detect a change in an attribute, and the receiving partition interprets this delay or lack of delay as information [9, p. 259].

For covert channels, Kemmerer [9, p. 259] identifies the following four preconditions:

1. The sending and receiving processes must have access to the same attribute of a shared resource.
2. (a) If it is a storage channel, there must be some means by which the sending process can force the attribute to change.
   (b) If it is a timing channel, then the sender must be capable of modulating the receiver's response time for detecting a change in the shared attribute.
3. If it is a timing channel, the sending and receiving processes must have access to a time reference such as a real-time clock.
4. There must be some mechanism for initiating the processes and for sequencing the events.

A strong partitioning mechanism that can be verified is access control to kernel resources (ACK).

Definition: access control to kernel resources means that each *kernel* resource is either completely separated by access control *or*, if it is reused, it is always separated by time windows. ACK is implemented by the separation kernel.

"Separated" here means that for each byte of the kernel state, the right to modify it is uniquely mapped to at most only one partition: if one was to "color" the kernel state, then each byte of memory that can be written via invocations from subjects from a partition would be uniquely assigned to one partition for the right to write and/or modulate it. For example if one has a green partition and a red partition, then (almost) each byte of the kernel state is either green or red, or uncolored. An example for the uncolored region is the schedule used by the cyclic scheduler (no partition can write to it).

Because the second precondition of Kemmerer is negated when ACK is reached, then timing channels are ruled out, as sending and receiving processes never have access to the same attribute of a shared resource.

For example, in the Fiasco.OS covert channel example given before, the shared resource where ACK has been violated would have been the memory cell(s) storing the number of mappings of a page already assigned, as in the example partition a_1 was able to write to it by creating 2047 for sending a "1" and less than 2047 mappings for sending a "0" and partition a_2 was able to read out information from the counter by attempting a mapping on its own.

Conversely, to give an example where ACK is preserved, that is also based on a counter, in [16] a counter for events (a communication mechanism) sent to individual partitions is uniquely assigned to a partition.

Moreover, to achieve the above-mentioned QM (Section 3.1), QM can be implemented by one kernel memory manager for *all* partitions and a quota for *each* partition, or by having one kernel memory manager for each partition. The second option is easier to review, and, as a beneficial side effect, it also avoids confidentiality violations via the reuse of memory for different partitions [2, Section II.A].

Access control to hardware resources (ACH). Shared hardware resources can potentially bring in information channels, typically covert channels by modulation of a hardware resource. For example, a hardware cache protects against direct attacks against the confidentiality of its content, but observing the cache's behavior can reveal information how other applications have used the cache. For example, imagine that application a_1 has used the cache, then gets scheduled away, then application a_2 uses the same cache. If application a_2 uses up all the cache it is potentially replacing some entries that had been previously created by application a_1. If applications a_1 and a_2 are in different partitions, the cache invocations have different address space arguments and by construction the cache ensures that a_1 cannot see the content of cache entries of a_2 and vice-versa. However, when then application a_1 gets scheduled again, its own use of the cache can detect whether many or few cache entries have been replaced, that is whether application a_2 was very active or not. For example, x86 systems usually have a single memory controller, which can then result in different latencies when concurrently addressed from more than one partition. Moreover, on multi-core x86, L2 and L3 caches are frequently shared between CPUs. Such cache attacks have been exploited also in virtualization contexts (e.g. [7, 10, 17]).

Definition: access control to hardware resources means that each *hardware* resource is either completely separated by access control *or*, if it is reused, it is always separated by time windows. ACH is implemented by the MILS core and its configuration.

Read-only accesses often do not introduce new channels. For example if a partition's application can find out their CPU affinity, but if the possible CPU affinities are predetermined by configuration, and it is to be assumed that communication is allowed between partitions that share the same time window as they can influence each other by modifying the data of priority-based scheduling, then being able to find out one's CPU affinity is not an additional covert channel.

Temporal normalization (TN). Reuse of resources is allowed, but the requirements are stricter than for safety: in safety (Section 3) WCET was established, but it was simply an upper bound.

In TN, not only a reused object is cleaned, but also the time that is used by another user, including the cleaning, is always the same, which is a stricter requirement than WCET.

Definition: TN is the reuse of an object registers, including appropriate initialization and termination, so that residual information is made unavailable on context switch and that the time for initialization and termination is always the same. TN is implemented by the separation kernel and its configuration. TN implies CR and WCET.

For instance, when the delay of a context switch is accounted to the ensuing partition, one way to make a WCET context switch (the previous example for CR) to TN could be to put in between a short empty time window without payload after a time window with confidential application payload.

5.2 Use of Partitioning Mechanisms

Resource	Partitioning mechanism(s)
User-space memory	ACU
CPU registers	TN
Kernel resources	ACK (implies QM and ACM)
Hardware resource	ACH
CPU time (latency)	CS + WCET

Table 4: Partitioning mechanisms for control of information flow between colluders

Table 4 summarizes the partitioning mechanisms discussed in this section and previous sections, to the extent they are used for control of information flow between colluders.

6. MULTI-CORE CONSIDERATIONS

6.1 Additional Relevant Partitioning Mechanisms

Drift avoidance (DA). Drift avoidance has to be implemented in order to synchronize the cyclical schedule at more than one CPU.

Definition: Drift avoidance is a mechanism that ensures that the drift between partitions running on different CPUs stays below a fixed bound during run-time of the system. DA is implemented by the separation kernel + hardware.

For example, on an Intel architecture, drift avoidance can be based on synchronization via IPI (Inter-Processor Interrupts).

6.2 Use of Partitioning Mechanism

CR DA can be used to go from single-core to multi-core.

For illustration Table 5 gives an overview of partitioning mechanisms that can be used for multi-core implementations that are based on the functionalities of ARINC 653 (Section 3) or "Safety + confidentiality by MMU" (Section 4).

Resource	Partitioning mechanism(s)
User-space memory	ACU
CPU registers	CR
Kernel resources	QM
CPU time (latency)	CS + DA + WCET

Table 5: Example of multi-core partitioning mechanisms: multicore safety

Multicore safety and security is a research-intensive area. Just adding one additional partitioning mechanism ("DA") might seem quite simple. However, for multicore also achieving the other partitioning mechanisms becomes more labor-intensive (like ACH and ACK), for instance multicore ACH has to ensure that for example memory controllers are also partitioned to avoid side-channels via a shared memory controller [10, Section IV].

7. SUMMARY OF SAFETY/SECURITY USE CASES, CLASSIFICATION ACCORDING TO SPACE AND TIME

Table 6 summarizes security use cases, and the mechanisms used to implement them. Moreover, the table contains a column assigning our mechanisms to "space" and "time".

Resource partitioning ensures that the partitioned resources are, at each time point, exclusively allocated to a resource partition. This can be implemented in the following two ways [3, Section 3.2.5]:

1. Resources can be used simultaneously, but are kept in different locations (for example, memory organized into segments, pages, or address spaces).
2. Resources at the same location are used by different resource partitions, but resource usage is in different time slices (for example access to a shared CPU).

ARINC-653 [1, p. 7] assigns the term "space partitioning" to (1) and "temporal partitioning" to (2). The distinction of space and time has become widely accepted (e.g. [18]). [8, Section 2.1] explicitly subsumes memory and I/O resources under spatial partitioning.

8. PERSPECTIVE FOR MILS SYSTEM USERS

We have already seen that some partitioning mechanisms are not only established by the separation kernel, but also by the MILS core (ACH, ACM, ACU, DA, WCET) and the configuration of the separation kernel (TN). While ACM, ACU, and DA are fixed after choice of CPU, and TN is defined by configuration of the separation kernel alone, ACH is not stable under additions to the MILS architecture beyond the MILS core itself.

For example for ACH, for (non-embedded) hardware in general, the notion of resource can be very broad and may require more fine-tuning, such as, in of-the-shelf PC workstations, the temperature of the CPU for which there may be

Safety/security use case	Section	Partitioning mechanisms implementing space separation	Partitioning mechanisms implementing time separation
Operating system protection	Section 2	ACM + ACU	CR
Real-time safety	Section 3	ACM + ACU + QM	CR + WCET
Safety + confidentiality by MMU	Section 4	ACM + ACU + QM	CR + WCET
Control of information flow between colluders	Section 5	ACK + ACU	TN
Multi-core	Section 6	Same as single-core	Single-core + DA

Table 6: Safety/security use cases and how they are implemented by partitioning mechanism: for example, to satisfy "real-time safety" it is necessary to implement ACM + ACU + CR + QM + WCET.

a readable sensor, or even if the sensor is not readable, the clock rate of the CPU which may be reduced because of the temperature, which may be detectable by the receiver. Another example are audible vibrations emitted by the computer that may be detected by the computer's own microphone.

9. RESULTS

We have mapped various concepts of partitioning to partitioning mechanisms implementing the partitioning in a MILS core. The main result is that different use cases of safety and security can be compared, and one can precisely talk about differences. If one is aware of differences, one can strive for and maintain strong notions of safety and security.

10. RELATED WORK

The least privilege abstraction specifies in a subject-to-resource policy in the separation kernel protection profile (SKPP)[6, 12, 13]. If such a least privilege abstraction is applied to all or all "significant" internal resources within the kernel [12, Section 5.1], the least privilege abstraction would be equivalent to the ACK partitioning mechanism. However, in SKPP itself [6, Section 2.3.2] the least privilege abstraction is used for specifying that subjects in a partition have heterogeneous requirements for access to exported (user-visible) resources.

Heitmeyer [5] for an unspecified separation kernel in 2008 identifies the following subproperties of separation: temporal separation, no-exfiltration, no-infiltration, separation of control, and kernel integrity. Temporal separation would be fulfilled by CR, separation of control would be fulfilled by CS. No-exfiltration, no-infiltration and separation of control would be fulfilled by ACK.

In terms of the Common Criteria for Information Technology Security [4], we think that the "real-time safety" use case which is realized by real-time safety separation kernels reflects the FDP_ACC (for writes), FDP_ACF (for writes), FMT_MTD, FRU_RSA class SFRs of the EURO-MILS protection profile draft. We think that "safety + MMU confidentiality", reflects the FDP_ACC (full, including reads, writes, and executes), FDP_ACF (full), FMT_MTD, FRU_RSA class SFRs of the EURO-MILS protection profile draft. In the protection profile draft, management data is called "shapes". In both SKPP and the EURO-MILS PP, the control of information flows is addressed by formulations of FDP_IFF.n where $n \geq 3$.

Laprie [11] identifies attributes of dependability (generically, for any system). From the idea, the decomposition of security mechanisms to achieve different use cases of safety and security (here: from general-purpose operating systems to MILS cores), is similar to the decomposition of "dependability" into different dependability "attributes" [11].

11. ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318353 (http://www.euro-mils.eu/). Discussions with EURO-MILS partners have been generally helpful for the ideas in this paper. Early versions of this paper have been discussed with Sergey Tverdyshev, Oto Havle, and Andrea Bastoni. The anonymous reviewers are thanked for numerous helpful observations and suggestions.

(Presented at the International Workshop on MILS: Architecture and Assurance for Secure Systems, at HiPEAC 2015 Conference, Amsterdam, 20 Jan 2015.)

12. REFERENCES

[1] *Airlines Electronic Engineering Committee (ARINC).* Avionics application software standard interface: ARINC specification 653. *Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, MD 21401, January 1997.* http://www.arinc.com/.

[2] *Christoph Baumann, Thorsten Bormer, Holger Blasum, and Sergey Tverdyshev. Proving memory separation in a microkernel by code level verification. In* Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on, *pages 25–32. IEEE, 2011.* http://www-wjp.cs.uni-saarland.de/publikationen/Baumann-AMICS2011.pdf.

[3] *Holger Blasum, Sergey Tverdyshev, Bruno Langenstein, Jonas Maebe, Bjorn De Sutter, Bertrand Leconte, Benoît Triquet, Kevin Müller, Michael Paulitsch, Axel Söding Freiherr von Blomberg, and Axel Tillequin. MILS architecture (euro-mils*

whitepaper), 2014. `http://www.euromils.eu/downloads/2014-EURO-MILS-MILS-Architecture-white-paper.pdf`.

[4] *Common Criteria Sponsoring Organizations. Common criteria for information technology security evaluation. version 3.1, revision 4, September 2012.* `http://www.commoncriteriaportal.org/cc/`.

[5] *Constance Heitmeyer, Myla Archer, Elizabeth Leonard, and John McLean. Applying formal methods to a certifiably secure software system.* IEEE Trans. Softw. Eng.*, 34(1):82–98, 2008.* `http://chacs.nrl.navy.mil/publications/CHACS/2008/2008heitmeyer-TSE.pdf`.

[6] *Information Assurance Directorate. U.S. government protection profile for separation kernels in environments requiring high robustness. Version 1.03, June 2007.* `http://www.niap-ccevs.org/cc-scheme/pp/pp_skpp_hr_v1.03/`.

[7] *Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-VM attack on AES. Technical Report 2014/235, 2014.* `http://eprint.iacr.org/`.

[8] *Robert Kaiser, Stephan Wagner, and Alexander Züpke. Safe and cooperative coexistence of a SoftPLC and Linux. In* 8th Real-Time Linux Workshop, Lanzhou*, 2007.*

[9] *Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels.* ACM Transactions on Computer Systems*, 1(3):256–277, 1983.*

[10] *Don Kuzhiyelil and Sergey Tverdyshev. Timing covert channel analysis on partitioned systems. In* Proceedings of escar Europe, Hamburg 18-19 Nov 2014*, 2014.* `https://www.escar.info/`.

[11] *Jean-Claude Laprie, editor.* Dependability: Basic Concepts and Terminology*. Springer, 1992.*

[12] *Timothy E. Levin, Cynthia E Irvine, and Thuy D. Nguyen. A least privilege model for static separation kernels. Technical Report NPS-CS-05-003, The Center for Information Systems Security Studies and Research, Oct 2004.* `http://cisr.nps.edu/downloads/techpubs/nps_cs_05_003.pdf`.

[13] *Timothy E. Levin, Thuy D. Nguyen, Cynthia E. Irvine, and Michael McEvilley. Separation Kernel Protection Profile revisited: Choices and rationale. In* Fourth Annual Layered Assurance Workshop (LAW 2010) Austin, TX, USA, 6-7 December 2010*. Applied Computer Security Associates, 2010.* `http://fm.csl.sri.com/LAW/2010/`.

[14] *Michael Peter, Jan Nordholz, Matthias Petschick, Janis Danisevskis, Julian Vetter, and Jean-Pierre Seifert. Undermining isolation through covert channels in the Fiasco.OC microkernel. Technical Report 2014/984, 2014.* `https://eprint.iacr.org/2014/984.pdf`.

[15] *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, December 2011.*

[16] *Freek Verbeek, Sergey Tverdyshev, Oto Havle, Holger Blasum, Bruno Langenstein, Werner Stephan, Yakoub Nemouchi, Abderrahmane Feliachi, Burkhart Wolff, and Julien Schmaltz. Formal specification of a generic separation kernel.* Archive of Formal Proofs*, 2014, 2014.* `http://afp.sourceforge.net/entries/CISC-Kernel.shtml`.

[17] *Michael Weiß, Benjamin Weggenmann, Moritz August, and Georg Sigl. On cache timing attacks considering multi-core aspects in virtualized embedded systems. In* 6th International Conference on Trustworthy Systems, Beijing*, 2014.*

[18] *Yongwang Zhao, Dianfu Ma, and Zhibin Yang. A survey on formal specification and verification of separation kernels.* Front. Comput. Sci.*, 2014.* `http://act.buaa.edu.cn/zhaoyw/research/survey2014.pdf`.