Response to "Partitioning Kernel Protection Profile"

FSMLabs White Paper.
Victor Yodaiken yodaiken@fsmlabs.com

Table of Contents

Introduction

This note is a quick response to the document "Partitioning Kernel Protection Profile Report"(PKPPR) by Mike Weller (Rockwell-Collins), Roger Odell (Rockwell-Collins) and Lee MacLaren (Boeing). The PKPPR was submitted for comment the to Real-Time and Embedded Forum of the OpenGroup as a possible standard for a RTOS security. FSMLabs is a real-time operating system developer and a vendor of real-time control software to primarily process control, manufacturing test, factory automation, instrumentation, and communication markets. While PKPPR has some strong technical merits, it is not appropriate for the more general control systems market, and we believe that it suffers from some technical and "threat analysis" drawbacks.

The fundamental technical premise of PKPR is solid. The heart of a secure system must be a trusted OS kernel that is small enough to be verified. But I want to focus here on areas of disagreement, rather than on the wide areas of agreement. The problem with PKPR is in (A) threat analysis and (B) over-simplified technical answer.

1. **Threat Analysis: PKPR ignores real-time.** The term, "real-time" is used in PKPR, but there is no discussion of scheduling or timing at all, except for a cursory warning about non-static scheduling algorithms (see below for more details). In fact, the core real-time requirements of Common Criteria for real-time resource control are passed over completely in PKPR. Threats that potentially change timing behavior are critical issues, so PKPR's decision to pass over priority and time allocation and to pay only the most cursory attention to resource allocation in general makes it unusable.
2. **Threat Analysis: PKPR passes over most of the Common Criteria functional requirements.** Indeed, PKPR reads like an "Orange Book" era specification in many respects. Of the 11 Common Criteria functional units, PKPR addresses only 4 and those 4 include FRU (resource allocation) mentioned above. All of the omitted or cursory functional units are important.
3. **PKPR reliance on a very simple use of memory protection hardware mechanisms rules out most of the RTOS market.** From low end "MMU-less" devices, to telecommunications systems using new generation network processors that include MMU-less microengines, to the dual DSP/Microprocessor systems also common in telecom, large servers where MMU usage is inherently complex, and finally multi-processor systems – none meet PKPR requirements. This is a huge part of the market and the areas that are seeing the most growth.
4. **PKPR requires that validating each component provides for compositional correctness**. This is simply incorrect in any but the simplest systems. The classical case of requiring that the span of security levels is bounded by some constant, is only one example.

Primary Threat Analysis for a Real-Time OS.

Common Criteria FRU vs. PKPR

The first security requirement of a real-time operating system is to make sure that nothing subverts the execution of critical tasks at the scheduled time. Common Criteria FRU specification defines three families, all critical to real-world real-time control systems.

> *The family Fault Tolerance provides protection against unavailability of capabilities caused by failure of the TOE. The family Priority of Service ensures that the resources will be allocated to the more important or time-critical tasks and cannot be monopolised by lower priority tasks. The Family Resource Allocation provides limits on the use of available resources, therefore preventing users from monopolizing the resources (Common Criteria).*

From our perspective, all three families essential and inter-dependent, but let's focus for now on the Priority of Service family which is at the heart of the matter.

> *The requirements of this family allow the TSF to control the use of resources within the TSC by users and subjects such that high priority activities within the TSC will always be accomplished without undue interference or delay caused by low priority activities (Common Criteria).*

PKPR does not mention Priority of Service but PKPR does comment on scheduling.

"Time is partitioned into time-fragments, which are allocated to each of the subjects. Time may be statically allocated (time-slice scheduling), or it may be allocated on demand based on preemptive priorities, deadlines, or some other scheduling principle. Each subject is allowed exclusive use and control of the CPU and related hardware during its time partition."

This specification appears to rule out use of DMA devices, interrupts (which conflict with "exclusive use") and multi-processor shared memory systems. PKPR goes on to state:

The choice of scheduling algorithm is not specified by this Protection Profile, but there is a caveat: non-static scheduling creates the possibility of covert timing channels and a denial-of-service threat. These risks should be clearly identified in the user guidance documentation. The accreditor of a system employing a partitioning kernel should consider the risks and mitigating factors if a non-static time allocation scheme is to be used."

I discussed this issue with MacLaren and we disagree. First, denial of service is a danger whether there are static or dynamic schedules in any practical system in the general control market. Why? Consider a device managing a machine tool or power switch that is connected to a network and suppose it is bombarded with spurious messages: the classical DOS attack. MacLaren argued that the solution is to simply not place such a system on the network, but this is not practical for the general control market. Everything is going to be on the Internet. We cannot pretend otherwise. Similar problems arise in any system that responds to interrupts or other events. From our point of view, instead of simply declaring this not to be a problem by fiat, we need to make sure that the RTOS is able to manage uncertainty and to bound delays caused by events.

Summary:

FRU is the most essential Common Criteria specification for a RTOS used in our markets. PKPP seeks to limit the complexity of offering FRU capabilities by limiting the environment in a way that is not feasible for all but some expensive custom and isolated applications.

Other Common Criteria issues for Control

I have not covered all 11 functional units in detail yet.

Multi-lateral versus Multi-level

PKPR has a focus on control and sanitization of information. These are the core issues of traditional "Orange Book" and other MLS systems and PKPR also has the traditional MLS focus on levels of security. While data integrity and security are critical issues, they are not the only critical issues and the hierarchical level model of secure-top-secure does not fit many control systems. The issue of "multilateral" as opposed to "multi-level" security is discussed in the literature (see, for example, Ross Anderson's standard textbook). A narrow MLS focus does not satisfy requirements for, say, a medical instrument which may contain confidential patient records or a telecommunications switch which needs to protect pricing data from unauthorized modification, but must make pricing data available.

FSMLabs takes the position that the RTOS cannot be an impediment to data security but cannot be the enforcer for more than a few basic rules.

Remaining functional units of Common Criteria

Class FAU: Security Audit

This class does not apply to the Partitioning Kernel, but surely it is needed.

> *Security auditing involves recognizing, recording, storing, and analysing information related to security relevant activities (i.e. activities controlled by the TSP). The resulting audit records can be examined to determine which security relevant activities took place and whom (which user) is responsible for them.(From CC)*

The RTOS must provide a mechanism for security audit trails or must accommodate a "plugin" offering this mechanism. For example, RTLinux ControlsKit is a RTOS provided standard interface to control variables and can be set up to trigger tracking, say, on commands that cause a value to cross a threshold or that attempt to change a protected variable.

Class FCO: Communication

This class does not apply to the Partitioning Kernel.

This class is concerned with "non repudiation". Imagine a factory automation system in which a control subsystem must validate the source of a command to be authorized. The non-repudiation of origin family provides a subject with a proof that a message comes from the claimed source. The non-repudiation of receipt family provides evidence that the data was received. This type of service is a common requirement in distributed control systems. As an example, RTLinux Lnet real-time networking can use the asynchronous mode behavior of IEEE 1394 to force hardware validation of non-repudiation assurances.

Class FCS: Cryptographic Support

This class does not apply to the Partitioning Kernel.

But there must be a provision in a secure RTOS for plugging in secure cryptographic support.

Class FDP: User Data Protection

*T*his is the Functional Spec that is most thoroughly investigated in PKPR. See above for the emphasis on multi-level vs. multilateral.

Class FIA: Identification and Authentication

This class does not apply to the Partitioning Kernel.

Class FMT: Security Management

Class FPR: Privacy

This class does not apply to the Partitioning Kernel.

Class FPT: Protection of the TSF

Covered in PKPR

Class FTA: TOE Access

This class does not apply to the Partitioning Kernel.

SUMMARY

Many of the functional units are not covered in PPKR, yet are important to a secure RTOS.

Limitations of the MMU

1. **a very large low end market segment.** The ARM7No-MMU is one of the worlds most widely utilized processors, it is at the heart of many control systems, and it is only one of many in a class that includes FR-V 440, ARM9No-MMU, Motorola DragonBall, and so on. None of these microprocessors have an MMU. Ruling these microprocessors out closes out a large part of a real-world market, needing a security profile.
2. **PKPR over-reliance on memory protection mechanisms rules out a very critical emerging market segment.** The Intel XSCALE processors, Texas Instrument Dual DSP/ARM, and many other new microprocessors utilize both a MMU capable microprocessor and a no-MMU "microengine" or DSP. Designs in which discrete DSPs are used in combination with microprocessors are also common. None of these meet PKPR assumptions.
3. **PKPR view of memory management is incompatible with high end server systems.** PKPR asserts that the low level Partitioning kernel must (1) be extremely small and simple(2) have sole control of the MMU and (3) prevent any use of shared memory. But the memory management mechanisms of modern server operating systems are not compatible with this model. The reason is that decisions on when to modify MMU can be very complex. Consider what happens on a page fault of a user process in an SMP server – and note that this could very well be a telecommunications switch using a PMC-Sierra dual RM7000 processor and running a SS7 database. The fault indicates an access to a logical page that is not mapped to a physical page in memory. The decision on when or if to modify the PTE depends on the I/O subsystem, the replacement algorithm, the underlying file system, and possibly on the capabilities and access control of the subject. Is the partitioning kernel going to know that process X has permission to map in block Y of file Z ? If so, it will be too big to be validated. If not, it will be placing responsibility for this decision in the hands of the external memory manager. In either case, the theory that a small, totally validated, partition kernel will handle the mapping is hard to credit. To me, this is a very interesting technical question that has several possible answers. In RTLinux, one answer may be that certain critical tasks operate in physically protected memory outside of the standard MMU control, and these tasks include components that periodically validate the integrity of the MMU control software in the larger system. There are several other paths, but the answer is not as easy as implied in PKPR.

Technical oversimplification: Composition.

To be completed. The theory in PKPR that one can trust composition of validated components seems misplaced and unjustified. A robust profile must discuss compositional validation.

The classical example is the when you compose a system that connects 2 levels of secure data to a component that connects the next second level to a third level, the result spans 3 levels, something that may be forbidden. More to the point, consider a system constructed by connecting multiple PLC parts to a small computer where each of the PLC parts has an insecure ethernet link and the task of the computer is to connect them in a useful way so that the composite system **is** secure.