DRAFT: Security for industrial real-time control

With notes on: ”Partitioning Kernel Protection Profile”

Victor Yodaiken
yodaiken@fsmlabs.com

Abstract

This note looks at security issues for real-time control systems and describes the RT-Core and RTLinux approach to security.

Contents

1 Introduction

FSMLabs develops and markets operating systems and related software for *infrastructure real-time control*: factory automation and manufacturing test, robotics, instrumentation, medical equipment, data communication and telecommunication, power transmission, and process control. Mission critical applications include infrastructure communication switches, the test frame for jet engines, satellite control, and machine tool controllers. Infrastructure real-time control software is becoming more pervasive and systems are becoming more sophisticated and more connected both to other control systems and to networks and databases. Because this type of software is pervasive, more and more of the infrastructure depends on it. Because the software is becoming more sophisticated, old old methods of manual over-ride are less effective for fail-stop security. And the traditional method of physical isolation is no longer practical as a security barrier.

This note describes FSMLabs evolving approach to security for control software based on our real-time operating system and control products. Our approach is within the Common Criteria framework [N2198] and is focused on the mission critical applications common to our customer base.

This note also responds to the document ”Partitioning Kernel Protection Profile Report”(PKPPR) by Mike Weller (Rockwell-Collins), Roger Odell (Rockwell-Collins) and Lee MacLaren (Boeing)[MWRC02]. PKPPR was submitted for comment to the Real-Time and Embedded Forum of the OpenGroup as a possible standard for RTOS security.

2 Security Goals and Design.

2.1 Vulnerability analysis

Any discussion of security must begin with a threat analysis.

1. *Economic barriers to use.* Security mechanisms that lower the economic value or decrease the utility or safety of software will be bypassed or not purchased. While it may be possible to mandate use of secure software in certain restricted military or highly safety critical civilian applications, such as core avionics systems, security measures must make economic sense to be of practical use in the larger infrastructure control market. See [And] for some valuable materials on this topic and particularly see [And01a].

2. *External Access.* Industrial control systems are often accessible over some network or via some other partially secured interface. External threats include denial of service attacks and unauthorized access. Note that the prescription to *put the system behind a firewall* does not solve the problem but merely states that there is a need to secure the communication channels.

3. *Real-time operation.* Real-time operating systems are much more vulnerable to ”denial of service” attacks than are time-sharing systems because a ”slow-down” on a time-sharing system may be a catastrophic failure on a real-time system. The Common Criteria Family Resource Usage (FRU) covers protection against attacks that break resource allocation including priority. This ”family” is at the core of a secure RTOS.

4. *Inherited Security Failures*. Industrial real-time control software will depend on commercial standard hardware and software. Exploitable security weaknesses in those components can produce holes in system security.

5. *Digital Rights Management Systems*. Commercial digital rights management (DRM) initiatives like the Trusted Computing Initiative embed dangerous trojan horses in both commercial software components and hardware platforms. DRM systems are both a point of entry for attackers and a source of possible *auto-immune* reactions where the DRM incorrectly detects license violations and damages system operation. See [Yod01] for more details.

2.2 Range of target systems and environments

Three example target systems should illustrate the nature of the challenge.

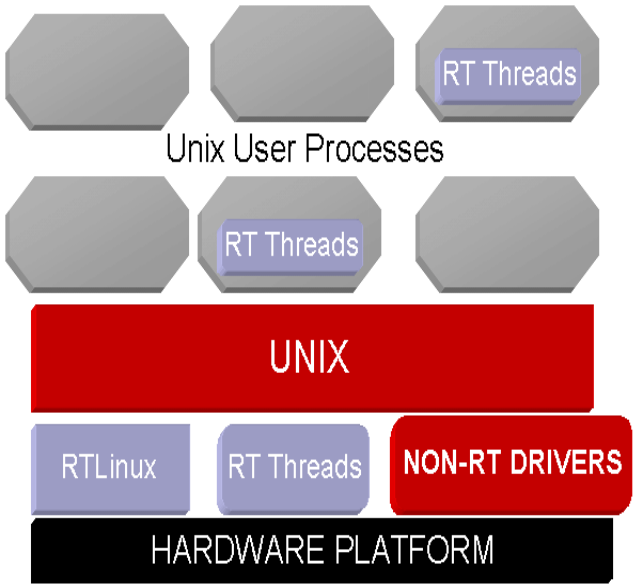
1. RTLinux is being used on a large scale ”machine-in-loop” test frame that runs on networks containing multiprocessor PC/Server type computers, VME backplanes, and several different Single Board Computers (SBCs). Timing requirements are on the order of 100 microseconds worst case scheduling jitter. The system is accessible via a corporate network supplying an operator interface and connection to databases. Data that is secret/Top-secret and company confidential is produced and used by this system. There are no untrusted users, but protection against external threats and trojan horses is essential.

2. RTCoreBSD is being used for a high speed network ”edge” device that routes voice and data streams. The hardware base is dual processor Pentium 4 machines that must also run large database programs. Timing requirements are on the order of 10 microseconds worst case jitter. Company and customer confidential data is stored on board. External threats are the primary concern.

3. RTLinux is used in a factory process control system consisting of a network attached to the Internet and corporate databases, and instances of control computers connected to both the local network and dedicated networks of simple controllers. The simple controllers are extremely cost and power sensitive and generally are small mmu-less devices with little additional memory to spare. The security of these simple components must be a composite property of their own protections and protections provided by the connecting node.

3 Basic Design Elements

3.1 RTCore Design



RTCore is a small hard real-time kernel that can run a secondary operating system as a pre-emptible thread. The common secondary operating systems are Linux (for RTLinux), and BSD UNIX. The intent of the design is to decouple timing critical operation from the complex services and applications of the non real-time system, but to make those services available to real-time software. RTCore applications are threads and signal handlers running either within a shared supervisor address space or within the address space of a host user process. As an indication of performance, a 1 millisecond real-time thread on RTLinux shows no more than 12 microseconds scheduling delay under heavy load on a standard AMD Athlon K7 based PC while the worst case scheduling jitter on a Compaq IPAQ PDA using a 200MHz StrongArm processor is under 35 microseconds.

3.2 Leveraging RTCore as a security monitor.

The basic RTCore design provides several avenues towards security using a *partitioning* approach similar to that used in [MWRC02]. RTCore imposes three partitions: real-time kernel, secondary system (kernel/user), real-time process space application. The secondary kernel and its applications provide a security system consisting of **P** (secondary kernel permissions), **N** (secondary kernel network and communications) **RTLock** (real-time kernel monitor), **LNet** (real-time kernel filter) and **CK** (controls kit) components. Heavyweight security components, base encryption, permissions/capability checking, IPSec, and other standard modules execute in the secondary kernel partition **P** and **N** components (possibly as user processes). The real-time kernel partition supports both the **RTLock** and **LNet** components. The **RTLock** component monitors the integrity of the **N,P** components. The **LNet** component is a real-time network system that *filters* communications. The **CK** component provides semantic dependent security. Many real-time applications export an operator and control interface in terms of sets of control variables that can be read and set (perhaps via a triggering operation or

by connecting functions to variables). We have developed a general purpose tool-kit called ControlsKit for exporting control variables in order to make it convenient to build graphical user interfaces, use browsers or even spreadsheets as operator interfaces and interconnect control systems in a structured manner. ControlsKit implements a rules-base for validating commands that modify or expose control variables and this rules base is the heart of the **CK** security methods.

1. The RTCore real-time kernel is small and amenable to verification. The full kernel, suitable for multi-processor operation is under 30,000 lines of code mostly divided into small independent modules. The RTCore kernel is especially well suited to enforcement of the Common Criteria FRU functional unit.

2. **RTLock** software can validate the integrity of the **P** and **N** components. Undetected compromise would require compromise of both systems under difficult timing constraints. For example, the security components of **P** can be periodically examined to validate that code pages remain read-only and not modified, that core data structures remain sensible and are correctly signed, that no unexpected process is running with too many privileges, and that security and networking components are running at reasonable frequency. The **RTLock** software might force a restart from secure storage if it detected an integrity failure or it might be required to periodically validate integrity to an external monitor so that any interference with its timing would be detected. This approach broadens the use of a RTCore security mechanism to enterprise and other systems that are not themselves time critical.

3. **LNet** partition software can provide a transparent network security level that can defend against denial-of-service and penetration attacks above the traditional protocol stacks and by detecting timing anomalies.

4. The **P** and **N** software in its own partition can run standard security and encryption modules, obviating requirements for expensive and poorly tested niche software.

5. When the target secure system interaction to the outside world is limited to going via ControlsKit the **CK** partition can do semantically rich analysis of commands and can require validation based on that knowledge base.

The standard RTCore method and this security approach can be applied to non-traditional secondary kernels, such as a Java Virtual Machine.

The approach taken here is specifically designed to not require hardware enforcement of secure address spaces but to be able to benefit if such hardware is present. The PKPR relies almost exclusively on hardware memory management units of standard microprocessors for separating domains. For reasons given in appendix A, we believe that such an approach has limited applicability in our domain. There are, however, some ways to use hardware to harden systems that we will actively explore.

1. **TCPA devices**. TCPA is a double edged sword for security. While, in its present incarnation TCPA seems more concerned with digital rights management (DRM) than with security, and thus actually creates security problems, sensibly designed TCPA hardware would provide secure encryption and secure key storage. Since this is the purported

aim of TCPA vendors, it is possible that enough customer resistance to intrusive DRM, usable TCPA hardware will become commonly available. This hardware will provide mechanisms for meeting Common Criteria FPT_SEP (separation of security domains) in a reliable fashion.

2. **External monitors.** If we either attach a hardware monitor or connect over a network to a monitoring node, we can provide hardware separation outside of the MMU. Suppose that we provide a device that can force reboot or shutdown and that requires a 1 millisecond periodic validation from the **RTLock** component. In this case, **RTLock** will do a lightweight and incremental check of **P** and **N** as well as its own integrity and will generate an appropriately difficult to counterfeit value to signal the monitoring device. To defeat this security, an attacker would need to compromise **P** and **RTLock**, and take over the validation task of **RTLock** *without producing a timing change*. Inactivating **RTLock** will not be enough to prevent a secure reboot. Instead of a special purpose device, we can use a peer or controlling node on a deterministic network. This second solution does not require any special hardware and is better suited to E-Commerce applications where detection of a possible compromised node can initiate a recovery or isolation operation.

3. **Special purpose memory protection.** The sophistication of standard paging MMUs is far beyond the requirements of separating security domains. A simple mechanism for protecting the data and instruction space the RTCore and **RTLock** component can easily be implemented outside the standard MMU.

3.3 High level view of security approach.

1. Defense in depth. As Schnier[Sch00] points out, we cannot rely on an unbreachable wall and must instead build a secure system to both be difficult to penetrate and to ”fail gracefully”. Our approach is to place as much as possible of the complex security system in secondary kernel and secondary kernel application code and to use the secure RTCore kernel to monitor the integrity of those components.

2. Partitioning. PKPPR identifies the complexity of secure kernel as a critical parameter. If the ”secure” kernel is too large and complex, it cannot be validated with much degree of assurance. We believe that the size of the RTCore kernel is not out of reach of thorough analysis and mathematical validation because the RTCore kernel is itself quite modular and structured. RTCore kernel is at a minimal size for a usable core kernel, for our application domain.

3. A range of profiles. We propose to define protection profiles for secured systems based on a two dimensional matrix of accessibility (‘ what can be guaranteed about the safety of the external environment) and event range (what inputs are permitted from the external environment). For accessibility: if a system is connected only to a dedicated serial cable that comes from a secure operator port, it needs less security than a system that has an Internet port or a phone line coming into it. For event range: if a system will ignore any inputs except for signed control packets of a fixed form, then it can be secured more easily than a system that accepts any IP packets.

(a) *Uncontrolled* for full Internet or phone or radio interface where a full range of threats are possible.

(b) *Plant network* for devices that are connected only to an intra-net or other partially controlled networks. Primary threats include unauthorized access, erroneous or spurious commands, data movement either to or from incorrect locations, and compromise of the plant network firewall.

(c) *Device Network* where controllers connect to a dedicated network that has only secure nodes and possibly a firewall to an external network. Threats here a primarily spurious commands, failures due to misconfiguration, and threats due to compromise of the firewall.

(a) *Unrestricted*. There are no *a priori* restrictions on what requests can be accepted and processed by the system.

(b) *Certified*. Only packets signed by a secure source are accepted.

(c) *Bounded*. There is a set of known commands and only commands from that that set are accepted. I assume that these will also be certified.

(d) *Fixed*. Bounded and also command/source pairs are specified in advance so it is known which sites can send what commands.

From the criteria defined, we can get a sense of the stress on the various security components. Table 1 shows the relationship between accessibility, permitted events, and security components. The top left cell has open communications and no restrictions on events, so it stresses all components. The bottom right cell is on a device and has a fixed set of events, so the **CK** component is the gateway for all requests and those requests can only come from safe sources.

EVENTS	ACCESSIBILITY		
	Open	Plant	Device
Unrestricted	All components are exposed to any attack	Same as Open case unless the Plant network firewall is to be relied on and we don't worry about internal accesses.	Minimize dependence on **N** since access is only from secure nodes, but still need **LNet** to screen out packets that come from outside and need **P,M** because we have no limit on request types.
Certified	**F,N,C** are more effective as an attacker must produce a fake certificate to get past them. **RTLock** can use semantic information from **CK**	Same as the Open case.	Minimize dependence on **P,N** since commands come from secure neighbors only and must be certified too.
Bounded	Limit dependence on **P** and on **N** if the bounds are well chosen. **CK** is much stronger.	Same as the Open case but perhaps we can relax **N** further.	Minimize dependence on **N,P**.
Fixed	**CK** is the primary security wall, but we need**LNet** to be the primary firewall and still need **N** to validate connections	Same as the Open case.	Minimize **N,P,M,F**. If **CK** works, we are safe.

Figure 1: Accessibility versus event range

3.4 Comparison with PKPPR and details

PKPPR specifies a protection profile for the memory mapped control section of a real-time opearating system. RTCore is agnostic about memory mapping. There is no reason why a PKPPR compliant partitioning kernel could not be used as the secondary kernel of a RTCore instantiation. PKPPR depends exclusively on memory mapping hardware to protect partitions from each other. For certain systems, our approach can benefit from memory protection for the real-time kernel partition, but we do not depend on it for two reasons. The first reason is that we must accommodate several classes of hardware platforms where the memory mapping hardware partition scheme is not practical. The second reason is that we believe that the use of real-time integrity monitoring of **P,N** by **RTLock** and of **RTLock** by an external monitor and the extensive use of the **CK** partition can provide a strong assurance without memory protection hardware. See appendix A for more details on memory management.

4 Common Criteria

The Common Criteria uses a large number of multi-layer three letter acronyms to disguise the clarity of the basic specification. The secured system we evaluate (Target of Evaluation or TOE) contains security components (TOE Security Functions or TSF) which enforce the security policy (TOE Security Policy or TSP) throughout the secured system (TOE Scope of Control or TSC). Communications with the secured system is, via the security function interface which is, surprisingly denoted by a *four* letter acronym:TSFI which, of course, stands for *TOE Security Functions Interface*. The security components must also manage a collection of *subjects* which are things like processes, and external objects that are local users, remote users, and remote devices (Remote IT Products). All of this is neatly summarized in figure 2.

For RTCore, there are 4 classes of subjects: Real-time application components fall into two categories: supervisor space and process space. Supervisor space real-time applications must be completely trusted and validated. Process space components run within the address space of UNIX processes are are prevented from uncontrolled

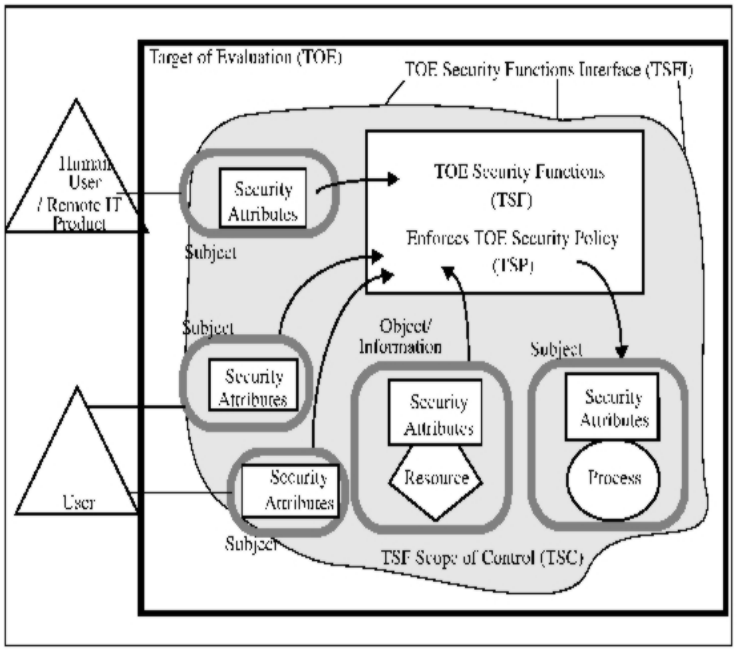


Figure 2: The Common Criteria Map of the World

access to the address space of the the RT-Core and can be prevented from monopolizing CPU time. Non-real time subjects include both the general purpose kernel and its objects such as file systems, and user processes.

Component	mode	supervision
RTCore kernel components	supervisor*	trusted
Kernel RT threads and handlers	supervisor	time
Process Space RT threads	user	all resources
Application kernel	supervisor	time, semantics
Application processes	user	all resources.

4.1 Class FAU: Security Audit

Security auditing involves recognizing, recording, storing, and analyzing 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)

We provide secure audit trail in **M,F,C** and use **RTLock** to ensure that **P,N** data is secure by, *e.g.* checking that the *inode* for the trace file never has write enabled by more than one task.

4.2 Class FCO: Communication

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.

Non-repudiation is application dependent and communication media dependent, but the RTOS and RTOS components need to provide non-repudiation services to match the application requirements.

LNet can force acks of raw messages that need this service and on some media, e.g. firewire, there is hardware support. The **N** and **P** components will need to have some non-repudiation support, but **CK** can offer a powerful non-repudiation system by using signed commands.

4.3 Class FCS: Cryptographic Support

There must be a provision in a secure RTOS for plugging in secure cryptographic support. This is straightforward, and we expect two levels of cryptographic support: a fast system in the RTCore and a possibly more sophisticated version in the secondary system. If components of the cryptographic system are placed within the secondary kernel or secondary kernel user processes, the RTCore security component can periodically validate the integrity of the cryptography system by: validating checksums on critical data structures, validating that the

physical pages have not changed and that code pages have not been updated since initialization of the security modules, and so on.

4.4 Class FDP: User Data Protection

This is mostly an issue of **RTLock** monitoring **P**, but it is worth noting that user data protection is often very complex in industrial real-time. In many of these systems we need ”multilateral” as opposed to ”multi-level” security. See, for example, Ross Anderson’s standard textbook[And01b] for discussion of this issue. A 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.

The **RTLock** component may ensure that critical files and code of the **P** and **N** components are not accessed by other users.

4.5 Class FIA: Identification and Authentication

> *Families in this class address the requirements for functions to establish and verify a claimed user identity.*
>
> *Identification and Authentication is required to ensure that users are associated with the proper security attributes (e.g. identity, groups, roles, security or integrity levels).*
>
> *The unambiguous identification of authorized users and the correct association of security attributes with users and subjects is critical to the enforcement of the intended security policies. The families in this class deal with determining and verifying the identity of users, determining their authority to interact with the TOE, and with the correct association of security attributes for each authorized user. Other classes of requirements (e.g. User Data Protection, Security Audit) are dependent upon correct identification and authentication of users in order to be effective.*

This is a **P,N** responsibility.

4.6 Class FMT: Security Management

The key issue here is definition of security roles.

4.7 Class FPR: Privacy

This is a **P,N** responsibility.

4.8 Class FPT: Protection of the TSF

> *This class contains families of functional requirements that relate to the integrity and management of the mechanisms that provide the TSF (independent of TSP-*

specifics) and to the integrity of TSF data (independent of the specific contents of the TSP data). In some sense, families in this class may appear to duplicate components in the FDP (User data protection) class; they may even be implemented using the same mechanisms. However, FDP focuses on user data protection, while FPT focuses on TSF data protection. In fact, components from the FPT class are necessary to provide requirements that the SFPs in the TOE cannot be tampered with or bypassed.

From the point of view of this class, there are three significant portions for the TSF:

- *a) The TSF's abstract machine, which is the virtual or physical machine upon which the specific TSF implementation under evaluation executes.*
- *b) The TSF's implementation, which executes on the abstract machine and implements the mechanisms that enforce the TSP.*
- *c) The TSF's data, which are the administrative databases that guide the enforcement of the TSP.*

(quote from [N2198])

4.9 Class FRU: TOE Resource Utilization

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 monopolized by lower priority tasks. The Family Resource Allocation provides limits on the use of available resources, therefore preventing users from monopolizing the resources ([N2198])

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[N2198]).

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. Everything is going to be on the Internet. We cannot pretend otherwise. But even without the Internet, any system that responds to interrupts or other asynchronous events must be DOS hardened. A secure RTOS must be able to manage uncertainty and to bound delays caused by events.

FRU is the most essential Common Criteria specification for a RTOS used in our markets.

4.10 Class FTA TOE Access

User access. For dedicated controllers, the only user access is an operator console or means of installing software. For more complex systems, there may be user access.

4.11 Class FTP: Trusted Path/channels

This is very much dependent on the accessibility parameter.

A Limitations of the MMU

- The non-mmu versions of the ARM7 are among the worlds most widely utilized processors, they are at the heart of many control systems, and this is only one of many in a class of mmu-less processors that includes FR-V 403, varieties of ARM9, Motorola DragonBall and so on. Ruling these microprocessors out closes out a large part of a real-world market, needing a security profile.

- The Intel XSCALE processors, Texas Instrument Dual DSP/ARM, and many other new microprocessors utilize both a MMU capable microprocessor and a no-MMU ”micro-engine” or DSP. Designs in which discrete DSPs are used in combination with microprocessors are also common. In all of these, the second microprocessor can bypass the MMU. PKPPR approaches require that these units be controlled outside the partitioning kernel by similarly validated software components.

- PKPPR authors rule out systems that have virtual memory. They need to do this because PKPPR requires that the 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 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.

B Composition

PKPPR stresses composability, which is indeed a key requirement for making provably secure systems. Composition is tough, however.

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

References

[And] Ross Anderson. Economics and security resource page. http://www.cl.cam.ac.uk/users/rja14/econsec.html.

[And01a] R. Anderson. Why information security is hard - an economic perspective, 2001.

[And01b] Ross Anderson. *Security Engineering*. 2001.

[MWRC02] Lee MacLaren (Boeing) Mike Weller (Rockwell-Collins), Roger Odell (Rockwell-Collins). Partitioning kernel protection profile report. Technical report, Submitted to RTEF, 2002.

[N2198] ISO/IEC SC27 N2162. *Common Criteria for Information Technology Security Evaluation - Part 2: Security Funtional Requirements*. 1998.

[Sch00] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, New York, NY, USA, 2000.

[Yod01] Victor Yodaiken. Digital rights management implications for safety and security. Technical report, 2001.