# High Assurance Security/Safety for Deeply Embedded, Real-time Systems

**R. William Beckwith**
Objective Interface Systems, Inc.
Herndon, Virginia, U.S.A.
bill.beckwith@ois.com

**W. Mark Vanfleet**
National Security Agency
Fort Meade, Maryland, U.S.A.
wvanflee@restarea.ncsc.mil

**Lee MacLaren**
Boeing Integrated Defense Systems
Seattle, Virginia, U.S.A.
lee.s.maclaren@f22.boeing.com

## ABSTRACT

*In this paper, we describe:*

*(i) a significant evolution to computer security architectures and secure communications MILS (Multiple Independent Levels of Security/Safety) capable of the high assurance to support MLS (Multi-Level Security) systems without the complexity of traditional MLS systems,*

*(ii) the MILS RTOS Partitioning Kernel architecture,*

*(iii) the MILS secure communications Partitioning Communications System architecture,*

*(iv) Real-time MILS CORBA, and*

*(v) industry efforts to provide implementations of this architecture.*

## KEYWORDS

MILS, security, high assurance, Common Criteria, EAL-7, Real-time CORBA, distributed systems

## SECURITY EVOLUTION

### Fail-First Patch-Later

Most commercial computer security architectures are a reactive result of problems that have resulted from insecure operating system and communications architectures.

This *fail-first patch-later* approach is **inappropriate** for the communications infrastructure supporting mission-critical telecommunications, data communications, utilities, transportation, aerospace, defense, financial, and similar mission-critical systems. The *fail-first patch-later* approach to computer security has interfered with the ability of intellectual property-based businesses such as the entertainment industry to guarantee product access and use control that is essential for recovering the large financial investments used to create these information products.

In addition, more and more critical infrastructure systems are accessible directly and indirectly through the Internet. The Washington Post reported that the FBI is concerned about the threat of terrorists attacking these critical infrastructure systems by leveraging flaws in computer security systems:

> U.S. analysts believe that by disabling or taking command of the floodgates in a dam, for example, or of substations handling 300,000 volts of electric power, an intruder could use virtual tools to destroy real-world lives and property. They surmise, with limited evidence, that al Qaeda aims to employ those techniques in synchrony with "kinetic weapons" such as explosives.

### Foundational Threats

Systems and application software is only as good as the foundation it is built upon. If malicious software can successfully attack the system's foundation can render almost any form of system or application security useless.

Foundational threats include:

- Bypass—malicious software circumvents the system's protection. If critical security software can be bypassed there is no assurance that application programs using security services are safe.
- Compromise—malicious software can read private data of other programs. If invasive software, like the spyware so common in today's Internet environment, can monitor the data of other programs then entire system security is suspect.
- Tamper—malicious software modifies the sensitive data of other programs. If tamper is possible then no application is safe from viruses, worms, etc.
- Cascade—malicious software causes failures to cascade from one system component to another. If a failure of one application can cause failure of another application it may be possible for much greater system failure. A notable example of unintentional failure cascade is a Navy cook who entered zero into a window that asked for a

number one to ten. The application divided by zero. This caused other applications failed. Eventually the O/S failed. The hard drive got screwed up. The system would not reboot. The ship was towed to shore.

- Covert Channel—malicious software that can leak information through a communication channel that is a side effect of the primary communication intent. For example, by detecting the presence or absence of a message an observer can derive information as to the activity of the communicating parties. If there are covert channels available a malicious communicating party can leak any information to the observing party by creating intentional timing messages in an arranged pattern.
- Virus—malicious software that runs at privileged levels so that it can infect all parts of the system and other systems. What is necessary is an architecture that enforces and manages the concept of least privilege. Then when a compromise occurs it damage is local, its damage can be detected, and recovered from. A big part of countering the computer virus problem is kicking device drivers and applications out of privilege mode.
- Subversion—malicious software is loaded by a user who thinks the software is legitimate. All code needs to be signed or it does not even load. The source of all software must be traceable to the original author. Software authors should follow good software engineering practices. Preventing subversion is everyone's responsibility.

The following example highlights the foundational threats in a notional architecture for a soft drink manufacturer:

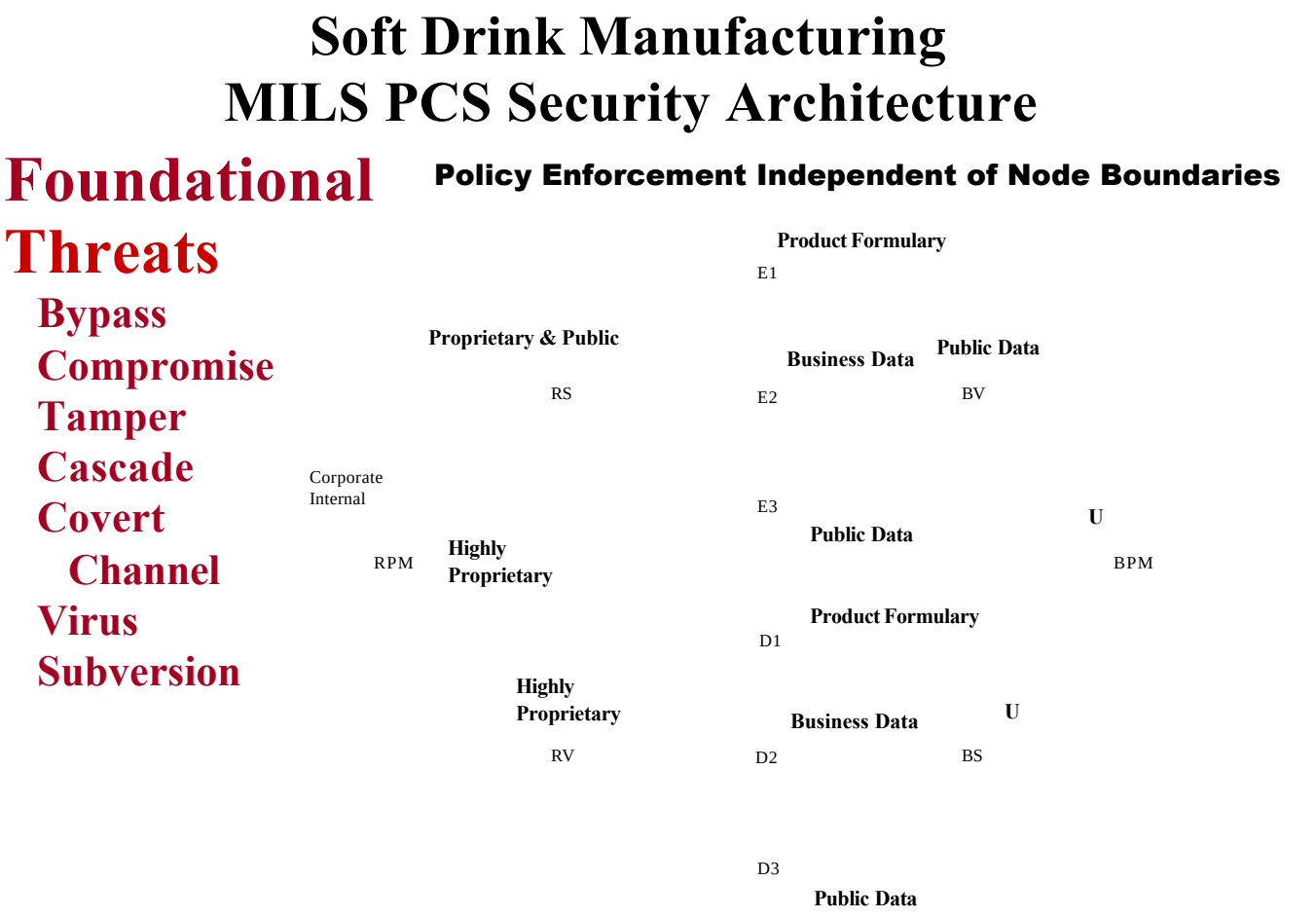


**Figure 1**

Successful soft drink manufacturers protect their product formulary above all else. A competitor who gains access to the formula to a successful drink can quickly steal market

share. Thus, the soft drink manufacturer requires that product formulary is kept separate from other data.

The problem is that without a strong foundation for guaranteeing that separation putting the product formulary on the same computers that have other business data and public data would subject their most prized secret to great peril from the bypass, compromise, tamper, cascade, covert channel, virus, and subversion threats. The soft drink manufacturer would be forced to rely on physical separation by prohibiting connecting the computers containing the product formulary to any other computers.

## Trusting the Foundation

An alternative to the *fail-first patch-later* approach is to use an approach designed to protect highly secure military systems. By *mathematically verifying* the core, trusted components of the operating system and communications system the potential for the system to fail its security objectives is dramatically reduced.

The history of past efforts to produce *mathematically verified* general purpose systems software is littered with commercial and financial failures. These efforts were all focused on the achieving the higher assurance levels in the U.S. DoD Orange Book.

The lower levels of security, in particular level C2, of the Orange Book were a wide commercial success in that C2 certification became a common requirement for banking, insurance, and other security conscious systems.

However, the Orange Book approach to high assurance systems fell short in two areas critical to modern secure systems software:

- The higher assurance levels (B3 and A1) required both *mathematical verification* of trusted system components and that those trusted systems components contain significant security functionality (MAC, DAC, auditing, et al) that made *mathematical verification* of those trusted system components virtually impossible.
- Intersystem communication was not addressed in the core security architecture of the Orange Book. These trusted components (and device drivers) typically all ran in privilege node in order to meet performance objectives in past years. Security critical application code also ran in privilege mode. This was a nightmare to evaluate. Such evaluations typically cost $100M.

## MILS

MILS (Multiple Independent Levels of Security/Safety) represents a relatively new (10 years) approach to building secure systems in contrast to the older Bell and LaPadula theories on secure systems that represent the foundational theories of the DoD Orange Book.

MILS makes *mathematical verification* possible for the core systems software by reducing the security functionality to four key security policies:

- Information Flow (between partitions both internally to a RTOS and end to end between partitions in different computing platforms, this requires authentication and integrity for end to end protection)
- Data Isolation (private data remains private, this may require encryption for end to end protection)
- Periods Processing (the microprocessor itself will not be a covert channel to leak classified data to unclassified processes as the processor move from partition to partition within a partitioning RTOS, to close covert channels on a communication link may require full period encryption, dummy traffic generators, etc).
- Damage Limitation (a failure in one partition will not cascade to another partition, failures will be detected, contained, and recovered from locally).

MILS requires that the *Partitioning Kernel* and the trusted components of *Middleware Services* are implemented so that the security capabilities have the following characterics:

- Non-bypassable (the security functions cannot be circumvented)
- Evaluatable (the security functions small enough and simple enough to be *mathematically verified* and evaluated)
- Always Invoked (the security functions are invoked each and every time)
- Tamperproof (subversive code cannot alter the function of the security functions by exhausting resources, overrunning buffers, or other forms of making the security software fail)

A convenient acronym for these characteristics is *NEAT*. The MILS architecture allows the creation of application code that is *NEAT*. While most MILS-based application code is freed from this level of rigor because it is protected from and limited from damaging other applications, some applications will need the highest level of assurance. These partitions with these applications will need to be *NEAT*. Such partitions are referred to as *Reference Monitors*.

The MILS architecture was developed to resolve the difficulty of certification of high assurance systems, by separating out the security mechanisms and concerns into manageable components. A MILS system isolates processes into partitions, which define a collection of data objects, code and system resources. These individual partitions can be evaluated separately, if the MILS architecture is implemented correctly. This divide and conquer approach exponentially reduces the proof effort for secure systems. To support these partitions the MILS architecture is divided into three layers:

- *Partitioning Kernel*—a very small (4,000 lines of code or less) *mathematically verified* piece of software

trusted to guarantee separation of time and space partitioning,

- *Middleware Services*—most of the traditional operating system functionality including device drivers and a *Partitioning Communications System* to extend the scope of the separation provided by the *Partitioning Kernel* to inter-system communication, and
- *Applications*—responsible for enforcing application layer security policies.

With most operating systems it is very difficult to prove that these requirements are being met. Recently, however, the MILS architecture has emerged in which a micro kernel is responsible for:

- partitioning the computer into separate address spaces and scheduling intervals,
- guaranteeing isolation of the partitions, and
- supporting carefully controlled communications among them.

Because that is all the kernel does, it can be very small – less than 4,000 lines of code. This makes it amenable to the formal analysis methods, comprehensive documentation, and exhaustive testing required for certification. The partitioning kernel design is also very helpful in achieving flight safety approval under DO-178B, and in fact that is where this development got started.

## PARTITIONING KERNEL

If we want to run a security function on the same computer as our application programs, where should it reside? In order to be tamper-proof, it must be in a separate address space from any un-trusted application code. And in order to be non-bypassable, it must be part of every input or output service request issued by an application. The natural solution, then, seems to be to put it in the operating system.

But mixing security functions and other code in the operating system's privileged kernel address space is less than optimal for two reasons:

- The security functions are often application-specific. And while many RTOSs are designed to allow some user customization, it would really be better not to have to make changes in the most sensitive part of the system.
- Since any code in the same address space as a security function could potentially interfere with the kernel's enforcement of security, the entire kernel must be analyzed for weaknesses and malicious code.

Enter the *Partitioning Kernel*. This is actually not new technology. John Rushby was describing the basic concepts at least as early as 1981. But it is new to the commercial RTOS world, where performance traditionally has trumped all other concerns.

Any operating system that supports multiple address spaces supports the concept of partitions and provides some measure of isolation, both between partitions, and between any partition and the OS. A partition requests an OS service by "trapping" to the kernel, i.e. by executing a special instruction that causes an interrupt. This puts the computer in kernel mode, which allows the OS to do whatever it needs to do, including reading or writing I/O control registers or even modifying the memory map. From here, the most efficient way for the OS to provide the service is to execute all of the service code in kernel mode. This saves context switches, but it places a lot of code in the kernel that could perhaps reside elsewhere.

What Rushby proposed was for the kernel to support carefully controlled communications between non-kernel partitions (Figure 2). The communication channels allow one partition to provide a service to another with minimal intervention by the kernel. RTOS services can thus be moved into non-privileged or partially privileged partitions, leaving behind only those functions that must execute in kernel mode. Figure 2 shows isolated partitions communicating through kernel-mediated channels. These channels are established statically when the partitions are created.

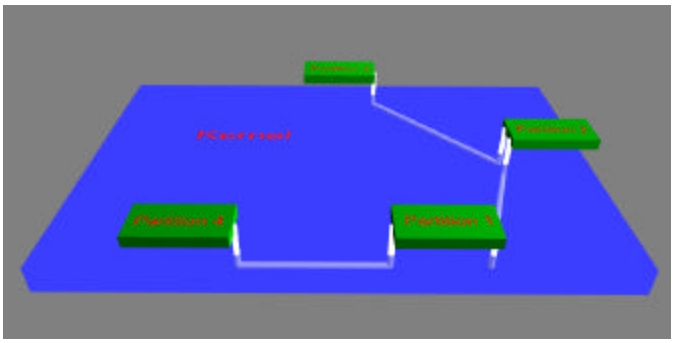

**Figure 2**

The goal of moving code out of the kernel is to make the kernel small enough to be verified by formal analysis and proof-of-correctness methods. A formal methods mathematician at the National Security Agency (NSA) has said that 1,000 lines of code would be ideal, but they would be willing to attempt something as large as 4,000 lines. The smallest commercial kernel is near the high end of that range.

Obviously, a system built on a partitioning kernel will suffer more context switching overhead than would occur in a more conventional design. This has been made more tolerable by very careful design of the inter-partition communication services, and also by hardware advances. In the current generation of PowerPC microprocessors for example, a full partition context switch can be completed in less than a microsecond. That means that 10,000 partition switches per second will consume less than one percent of the processor throughput. Partitioning is not free, but the cost has become much more tolerable.

So a practical, high-assurance, real time partitioning kernel is now within reach. This kernel guarantees that partitions are isolated from each other, and that only explicitly authorized communication occurs among them.

### Safety and Security

The military market for high-assurance security systems alone has not been large enough for the commercial RTOS vendors to justify investing in expensive evaluations. However, the commercial avionics market has attracted their investment dollars, especially with the imminent adoption of Global Air Traffic Management (GATM) rules.

The ARINC-653 standard was written specifically for avionics computing, especially where safety of flight was a concern. It specifies an RTOS design very much like the partitioning kernel just described, and with exactly the same goal: to allow two or more programs to share a computer while guaranteeing that they cannot interfere with each other. In an ARINC-653 system, both memory and processing time are statically allocated to partitions using configuration tables. A static network of communication channels is also established among the partitions. With the exception of a few kernel services, such as reading the real time clock, all input and output for a partition go through these channels.

ARINC-653 specifies a generic framework for enforcing an application-specific information flow control security policy. Information can flow from one partition to another only in the ways specified in the static configuration tables, and the partitioning kernel guarantees that this is so.

### Middleware Services

The middleware services layer provides for an extended scope of the separation concepts introduced by the partitioner. Middleware services are concerned about end-to-end data processing, and not just the single microprocessor data processing of the *Partitioning Kernel*. At the middleware layer, we begin to enforce the more traditional concepts of information flow. Each partition/address space in the system, no matter which microprocessor it is resident on, has a unique security label/classification. The system architect uses these labels to define the authorized communication between components. The labeling of the partitions and communication channels is used to satisfy the security policy. The middleware level is responsible for ensuring end-to-end security, through the following:

1. Labeling. The middleware layer must ensure that messages sent between individual partitions are correctly labeled with the sender's security classification.
2. Filtering. The middleware layer is responsible for filtering out any messages that are not appropriately labeled before delivering them to the recipient.
3. Maintaining Information Flow Controls. The system architect designs the system with specific authorized information flow restrictions, and it is these restrictions that the middleware layer enforces.

At the middleware layer we can introduce the concept of authorized information flow. If the system architect designs the system so that two partitions can communicate, then information flow between these partitions is authorized. A system can be designed to be a collection of isolated enclaves, where partitions exist within a single enclave and there is no information flow between enclaves. The MILS architecture will now allow the use of computer security measures to build systems and achieve the same assurance levels as these "physically isolated component" systems.

In the MILS architecture, all O/S code not necessary for performing *Partitioning Kernel* functions was moved out of privileged mode. Thus, by default, O/S service code (e.g. device drivers, file system, POSIX) has moved into the middleware layer. This is done to prevent various software and network attacks from elevating a processes privilege to an unauthorized level. Each enclave's partition can be configured to utilize a single set of O/S services code with it evaluated according to the enclave's requirements.

### Application Layer

MILS empowers the application layer to protect itself. And the application layer is responsible for enforcing application security policies. It is at this layer that the system provides for application-specific security policies. Any partition that processes data from more than one secure application realms must be considered a privileged partition.

## PARTITIONING COMMUNICATIONS SYSTEM

The *Partitioning Communication System* is a portion of MILS Middleware responsible for all communication between MILS notes.

The purpose of the *PCS* is to extend the protected environment of MILS kernel to multiple nodes. The *PCS* was developed with a similar minimalist philosophy to MILS Partitioning Kernel.

The *PCS* extends the four MILS policies to include end-to-end versions of these policies, but only these policies.

- *End-to-End* Information Flow
- *End-to-End* Data Isolation
- *End-to-End* Periods Processing
- *End-to-End* Damage Limitation

Thus, the *PCS* Leverages the MILS Partition Kernel to enable the application layer entities to enforce, manage, and control their own application level security policies in such a manner that the application level security policies are non-bypassable, evaluatable, always-invoked, and tamper-proof.

The result is a communications architecture that allows the security kernel and the *PCS* to share the responsibility of security with the application.

PCS must provide the following capabilities:

- Strong identity of nodes within enclave
- Cryptographic separation of levels
- Cryptographic separation of communities of interest
- Bandwidth provisioning & partitioning
- Secure configuration of all nodes in enclave
- Secure application instantiation
- Secure clock synchronization
- Signed partition images
- Elimination of covert channels (both storage and timing)

## REAL-TIME MILS CORBA

The synthesis of MILS and Real-time CORBA yield an unexpected benefit. The flexibility of Real-time CORBA allows easier realization of MILS protection.

The MILS architecture is all about location awareness. A properly designed secure system built with MILS will by intent separate appropriate functions into separate partitions to take advantage of the MILS partitioning protection.

Real-time CORBA is all about location transparency. The application code of a properly designed distributed system built with Real-time CORBA will not be aware of the location of the different parts of the system. This great flexibility can be used to optimize performance by rearranging what partitions each system object executes in. Mistakes in system layout can be corrected late in the development cycle.

The combination of MILS with Real-time CORBA allows system engineers to rearrange the location of system functions in order to better take advantage of the MILS partitioning protection.

## MILS CAN HANDLE MLS

While a MILS *Partitioning Kernel* is quite ignorant of the traditional Multi-Level Security (MLS) required for military and intelligence systems MILS is quite capable of supporting MLS systems. MILS can be used to construct such systems because of the strong separation guarantees the MILS architecture and certification process.

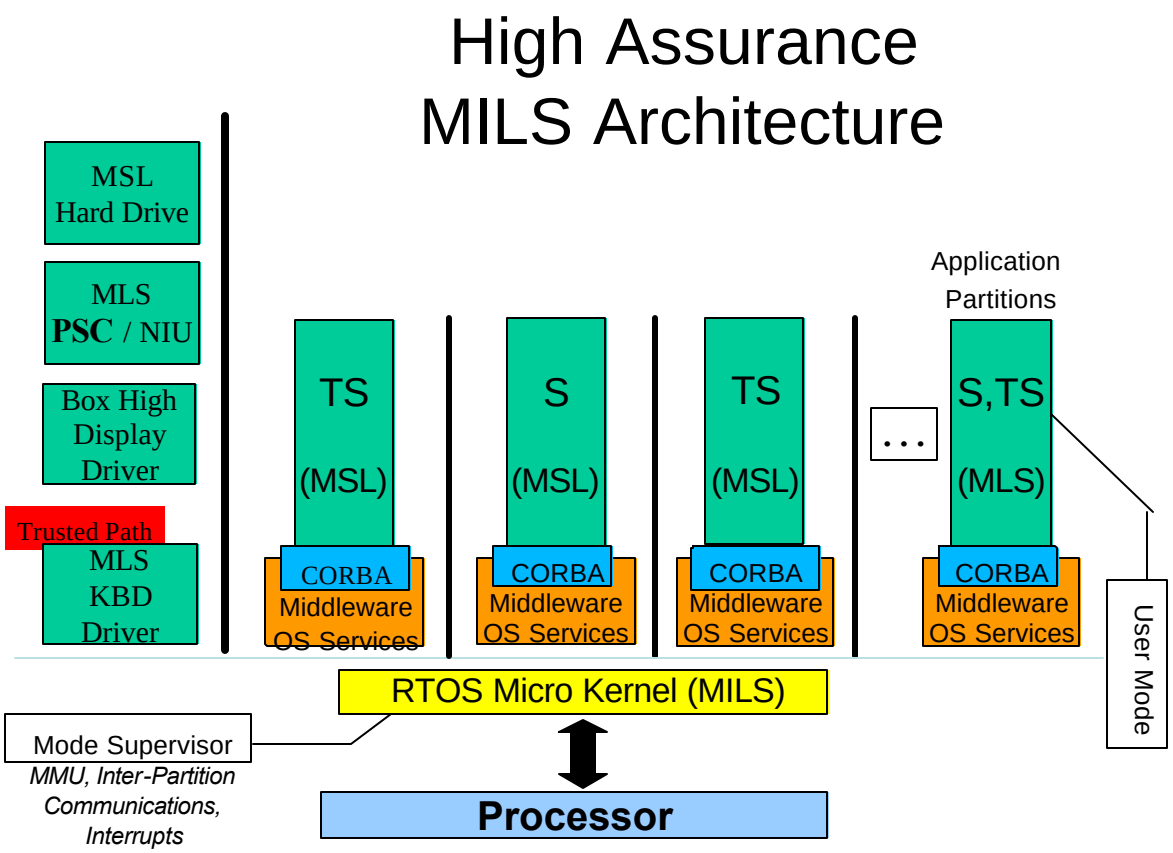


**Figure 3**

Figure 3 shows the realization of a multi-level security system using the MILS architecture. The green blocks are user level partitions. Only the Micro Kernel (the *Partitioning Kernel*) can access any privileged hardware capabilities. Accordingly the device drivers are all shown executing outside of system privilege because they are not in the micro kernel (*Partitioning Kernel*) address space.

## INDUSTRY SUPPORT

At least three commercial RTOS vendors either have built, or are in the process of building, MILS-compliant operating systems:

- Green Hills Software, Inc.
- LynuxWorks, Inc.
- Wind River Systems, Inc.

The U. S. Air Force, Boeing, Lockheed-Martin, Objective Interface Systems, Rockwell Collins, University of Idaho, and the National Security Agency are partnered in the effort to integrate several MILS security separation kernels with a Real-time CORBA middleware implementation. The results of this effort should support an OMG standardization effort for high assurance Real-time MILS CORBA.

## REFERENCES


[1] R. W. Beckwith, E. D. Jensen, D. Allen, et al, *Real-time CORBA 2.0: Dynamic Scheduling Final Adopted Specification*, Object Management Group, ptc/01-08-34 (Aug 2001).

[2] D. E. Bell & L. J. LaPadula, *Secure Computer Systems*, Technical Report ESD TR-75-306, MITRE Corporation, (Mar 1976).

[3] W. R. Bevier, "KIT: A Study in Operating System Verification", *Technical Report 28*, Computational Logics Inc. (Aug 1988).

[4] J. Currey, R. W. Beckwith, et al, *Real-time CORBA 1.1 Specification*, Object Management Group, formal/02-08-02 (Aug 2002).

[5] *Common Criteria for Information Technology Security Evaluation*, Version 2.1 (19 Sep 2000).

[6] *Department of Defense Trusted Computer System Evaluation Criteria* (The Orange Book), DoD 5200.28-STD

[7] E. Dube, and M. Weller, "A Path to Multiple Levels of Security", *Proceedings of the Information Assurance Solutions Working Symposium* (Dec 2000).

[8] J. W. Freeman, R. B. Neely, & M. A. Heckard, *A Validated Security Policy Modeling Approach* (May 1994).

[9] J. A. Goguen and J. Meseguer, "Inference Control and Unwinding," *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, (Apr 1984).

[10] K. Loepere, "OSF Mach Kernel Principles, Final Draft", Open Software Foundation and Carnegie Mellon University, (3 May 1993).

[11] R. Nelson, GTE, NSA R2 Research Contract (1994).

[12] Objective Interface Systems, Inc., ORB*express* RT, http://www.ois.com.

[13] R. W. Olsen & E. E. Cale, Real-time CORBA Trade Study, "Volume 6 – 2001 Supplement", Boeing Phantom Works Advanced Information Systems (9-5430), Document Number D204-31159-6 (22 Jun 2001).

[14] J. Rushby, The Design and Verification of Secure Systems. 8th ACM Symposium on Operating System Principles, December 1981. ACM Operating Systems Review, Vol. 15, No 5.

[15] J. Rushby, "Noninterference, Transitivity, and Channel-Control Security Policies," *SRI Computer Science Laboratory Technical Report* CSL-92-02 (Dec 1992).

[16] J. Rushby, Stanford Research Institute, NSA R2 Separation Kernel Conference (1994).

[17] J. Rushby, "A Trusted Computing Base for Embedded Systems," Proceedings of the 7th Department of Defense/NBS Computer Security Conference, pp. 294-311 (Sep 1984).

[18] W. M. Vanfleet, Kernel, *Middleware, and Application Level Security Policy Guidance For Deeply Embedded Systems* (Jun 2000).

[19] W. M. Vanfleet, *Designing INFOSEC Equipment with Separation Kernels*, Design Guidance INFOSEC Equipment Seminar ND-285 (Apr 2002).

[20] M. Weller & W. M. Vanfleet, *Computer Security In The Joint Tactical Radio*, MILCOM 2001.

[21] B. Gellman, "Cyber-Attacks by Al Qaeda Feared, Terrorists at Threshold of Using Internet as Tool of Bloodshed, Experts Say," *Washington Post*, Page A01, (June 27th, 2002).

[22] *IATF Release 3.1*, National Security Agency, (September 2002).

[23] *Partitioning Kernel Protection Profile (PKPP)*, (May 2003).

[24] J. Alves-Foss, The Multiple Independent Levels of Security/Safety (MILS) Architecture, Personal Notes

[25] *Common Criteria for Information Technology Security Evaluation*, Version 2.1, (August 1999).

[26] J.M. Rushby, "Proof of Separability: A verification technique for a class of security kernels", *Proc. Inter-*

*national Symposium on Programming, Lecture Notes in Computer Science*, 137:352–367, (1982).

[27] W.M. Vanfleet, et al, *An Architecture for Deeply Embedded, Provable High Assurance Applications,* (May 2003).

[28] M. Paulk, C. Weber, S. Garcia, M. Chrissis, M. Bush, *Key Practices of the Capability Maturity Model* , Version 1.1, Technical Report CM