A Multi-Layered Approach to Security in High Assurance Systems.

Conference Paper · January 2004

DOI: 10.1109/HICSS.2004.1265709 · Source: DBLP

3 authors, including:

Jim Alves-Foss
University of Idaho
150 PUBLICATIONS **1,479** CITATIONS

SEE PROFILE

Paul Oman
University of Idaho
112 PUBLICATIONS **2,272** CITATIONS

SEE PROFILE

A Multi-layered Approach to Security in High Assurance Systems[1]

Jim Alves-Foss, Carol Taylor, and Paul Oman
Center for Secure and Dependable Systems
University of Idaho
[jimaf,ctaylor,oman]@cs.uidaho.edu

Abstract

Past efforts at designing and implementing ultra high assurance systems for government security and safety have centered on the concept of a monolithic security kernel responsible for a system-wide security policy. This approach leads to inflexible, overly complex operating systems that are too large to evaluate at the highest assurance levels (e.g., Common Criteria EAL 5 and above). We describe a new multi-layered approach to the design and verification of embedded trustworthy systems that is currently being used in the implementation of real time, embedded applications. The framework supports multiple levels of safety and multiple levels of security, based on the principle of creating separate layers of responsibility and control, with each layer responsible for enforcing its own security policy.

1. Introduction

Over the past 25 years, numerous attempts have been made to engineer high assurance computer systems both for security and safety. Many of these systems were based on the concept of a monolithic security kernel whose main purpose was to oversee security for the entire system [0]. The intent was to have the kernel provide the basic security features and functionality needed for the system. In addition to a security kernel, there was the Trusted Computing Base (TCB) consisting of all of the security functions that assisted with system security. In practice, the TCB consisted of any driver, component or application that had any security decision making or security maintaining functionality. Unfortunately, systems based on a security kernel were generally not successful. The problem with security kernels is that application specific functions find their way into the kernel and into the TCB so that the TCB grows to eventually consume the whole system. Consequently, evaluation of these systems becomes extremely difficult due to the size and complexity of the security related code. In particular, several past DOD sponsored projects, ISA/AMPE, BLACKER, and CANEWARE cost nearly half a billion dollars and were eventually terminated prior to production. All were based on the security kernel concept [21].

Enforcement of security via a monolithic security kernel appears even less promising in the face of today's larger, more distributed systems. In this paper, we utilize a multi-layered approach to the design and verification of high assurance systems. The targeted environment for these systems is the real-time, embedded world of avionics and military multi-level applications. Systems in these environments must support communications between diverse groups with different clearance levels. We use a hierarchical system architecture, where multiple layers provide specific, well-defined security mechanisms that can be used by higher levels. When a system is designed to provide a security mechanism, the mechanism must be: (i) always invoked, (ii) non-bypassable, (iii) tamperproof, and (iv) verifiable. These properties embody the traditional *reference monitor* approach of past high assurance security systems [13].

The layered architecture consists of a partitioning real-time kernel, secure middleware, and a restricted I/O application layer. The approach is based on the concept of *separation* defined by John Rushby [15] in the early 80's. Separation requires independent processing units to execute in separate partitions (address spaces). Although separation provides for a high level of security, the performance impact of the large numbers of context switches has been prohibitive until recently. With modern processor speeds and capabilities, we now have the capability of implementing fully partitioning systems. The ARINC-653 standard [3] requires partitioning. Operating

[1] This work partially funded by Lockheed Martin Aero.

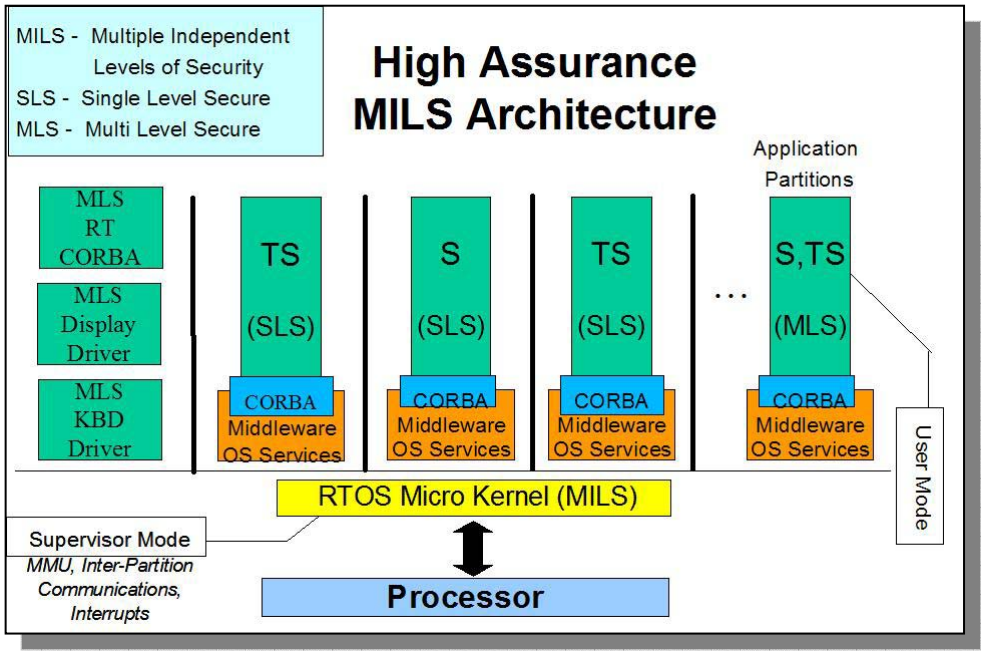

Figure 1. MILS Architecture

systems such as Greenhill's *Integrity* and LinuxWorks *LynxOS* provide support for full partitioning [2].

Given these recent advances the concept of separation through partitioning has been recently expanded [1,16]. Through separation, we can now develop a hierarchy of security services, where each level uses the security services of a lower level to provide new security functionality. Each level is responsible for its own security domain and nothing else. Limiting the scope and complexity of the security mechanisms provides us with manageable and, more importantly, evaluatable implementations. The architecture has been named, Multiple Independent Levels of Security (MILS)[2].

The security policy enforced by the architecture is based on *information flow*, *data isolation*, *periods processing*, and *damage limitation*. Information flow ensures that only authorized communication can occur. Data isolation protects data segments from corruption by unauthorized entities. Periods processing helps enforce information flow and data isolation by sanitizing shared resources such as processor registers between context switches. Damage limitation guarantees that a failure in unevaluated code does not compromise the continued processing of critical applications.

A key concept presented in this paper is the idea of layered separation in the enforcement of the security policies and in the evaluation of the high assurance components. The focus of this architecture is to allow the layers, as depicted in Figure 1, to cooperate in providing security. Each layer is responsible for security decisions at the appropriate level. The evaluation of MILS compliant products will focus on the use of the features of the lower layers to support the security decisions of the component. As an example of such products we discuss Common Criteria Protection Profiles [9] for both the Partitioning Kernel (PK) and MILS CORBA, which is a subset of our middleware layer. We show how a layered architecture approach can be used to implement a secure middleware service on top of a separation kernel and discuss verification issues for a Common Criteria evaluation at EAL5 and above.

The remainder of this paper provides details of this approach. In Section 2 we present the multi-layer architecture. In Section 3 we discuss the security policy behind the architecture. In Section 4 we discuss Common Criteria certification of MILS. In Section 5 we present other approaches to high assurance development, and in Section 6 we conclude the paper and outline current and future work.

[2] The MILS Architecture is a collaborative effort between government DOD agencies, industry and several universities.

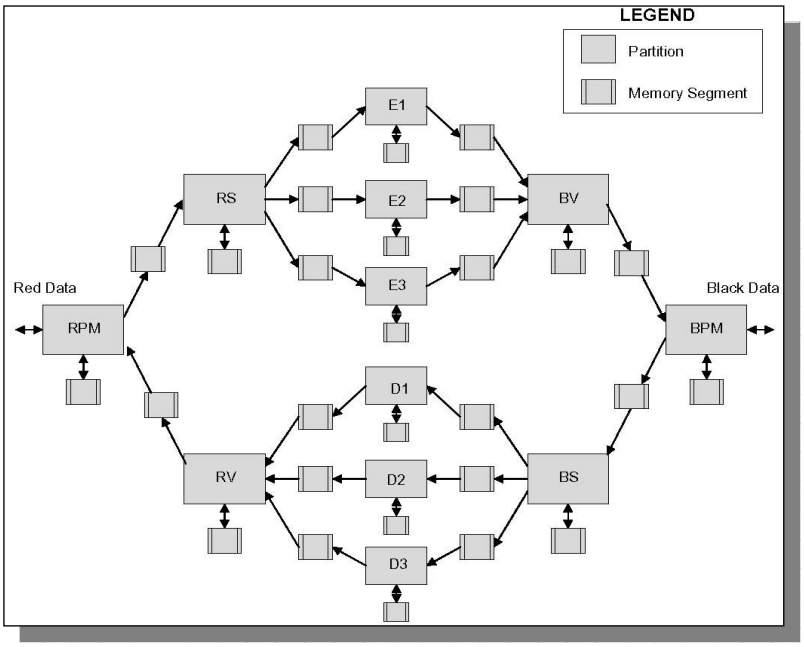

Figure 2. System Graph for Shared Memory

2. MILS Layered Architecture

The MILS architecture is intended for use in specific high assurance environments. As such, we have incorporated high assurance requirements for the system including:

- Support for Multi-level Security (MLS)
- Verifiability at a high assurance level
- *reference monitor* functionality at each layer

A secure multi-level system is one in which the system provides mechanisms to enforce mandated controls on the flow of information between processes executing at different security levels. A system that supports MLS security is one where the system tags objects with a classification level, tags processes with a clearance, and ensures that the data is manipulated by the processes according to the security policy.

Verifiability requires certification by an outside authority, which, in turn, requires that the system use easily understood security mechanisms that are simple and modular.

For the system to implement reference monitor functionality simply means it employs a mechanism that is always invoked, non-bypassable, tamperproof, and evaluatable. The concept of a layered approach to secure system design is not new. Neuman [10] described how layers reduce the risks of system failures and compromise compared to monolithic architectures.

2.1 Partitioning Kernel Layer

At the base of the MILS Architecture is the Partitioning Kernel (PK). The basic idea of the PK is to separate processes, or tasks, on a single processor into multiple processing engines (partitions) separated in both space and time. Each partition appears to have its own dedicated processor and operating system (Figure 1). As shown in Figure 1, some partitions will be designated Single Level Secure (SLS), consisting of a single data classification while others will be MLS supporting several data classifications. Instructions and data are accessed via a memory map that is controlled by the PK. All partitions are separated in time through the partition scheduler in the kernel. Time is divided into intervals, which are allocated to each process. The scheduling algorithm can be static as in time-slice scheduling, or dynamic as in preemptive priority or other demand-based scheme. During its time interval, each partition has exclusive use of the CPU and related system resources; but it is restricted in how it can interact with other partitions.

Separation in space is accomplished through strict partitioning of the memory into process specific areas. Only one process at a time ever has read/write access to a given area of memory. Consequently, no two processes are associated with the same memory for read/write access. A limited form of shared memory is allowed where two processes can share read access such as a shared memory library. Another form of shared read/write access is allowed under the

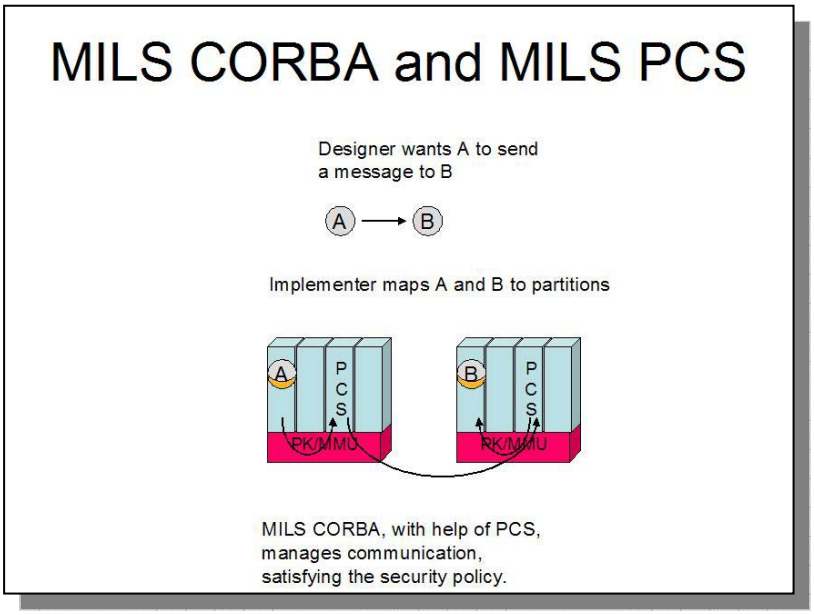

Figure 3. Deployment of MILS CORBA

assumption that an external protocol (i.e., a security arbitrator) oversees the sharing.

At this level, the system architect views the system as a collection of execution engines, each of which processes data at one or more security levels, and limited communication channels that provide for information flow between the partitions. These channels may be shared memory segments, or kernel-supported data streams. This view of the system can be depicted as a directed graph with vertices for the partitions and edges for the channels. For a shared memory system, for example, the system can be viewed as the directed graph depicted in Figure 2, where partitions are only connected to memory segments and memory segments are only connected to partitions. A partition can read or write a connected memory segment depending on the direction of the edges.

The PK is a minimalist approach to ensuring multiple levels of safety and security. It provides high assurance partitioning and information flow only, with other operating system functions such as device drivers, file managers, and message-passing services implemented outside the kernel in the middleware layer. A description of a PK for use in certification is currently being developed [14].

2.2 MILS Middleware Layer

The second layer in the MILS Architecture is the middleware layer that logically sits on top of the PK (as shown in Figure 1). In this architecture, middleware, should not be confused with the common definition of middleware within the software development community (i.e. OMG CORBA or Microsoft COM). Here, the concept of middleware includes a secure version of CORBA, which we refer to as MILS CORBA, plus other operating system services excluded from the PK such as file systems, device drivers, and network services. Where, in a traditional operating system, these services may execute in user space, or kernel space, in the MILS architecture we may have to segment these services where a portion of the service will now reside in a separate partition and the rest in the user's partition. MILS middleware will never reside in the kernel. While the MILS middleware layer extends the middleware services, CORBA still plays a significant role. A brief description of CORBA is provided below.

CORBA stands for Common Object Request Broker Architecture and was developed by an industry consortium known as the Object Management Group (OMG) in 1989 [5]. CORBA is essentially a distributed client-server message-passing system based on object oriented computing [5]. CORBA's greatest strength is its interoperability between heterogeneous computing environments, allowing programmers to write client-server applications while ignoring the low-level network details. The central component of CORBA is the Object Request Broker (ORB) which includes all of the communication infrastructure between clients and objects. The ORB handles object identity and location, connection management, and data delivery [8].

CORBA has traditionally been used in large enterprise systems where multi-platform computing is common [5]. CORBA's use in high assurance systems is relatively new and security features were only recently added in 1999 [12]. CORBA security resides with the ORB and exists in the application process space. Thus, security mechanisms are easily modified by other software objects within this process space [12]. Security is bypassable under this scheme and thus violates the reference monitor requirement for MILS middleware. Another problem with using existing CORBA implementations is that CORBA does not handle multi-level communication, which is another requirement of our MILS architecture.

Since CORBA security will not meet our needs for a reference monitor middleware mechanism, security for MILS CORBA is based on an alternate approach involving an additional component, a Partitioning Communication System (PCS), based on the concept of a Trusted Network Interface Unit (TNIU), originally proposed by Rushby and Randell [19]. The PCS has the important responsibility of labeling and exporting the user classification of each user mode partition with every user data stream. At least one PCS will be associated with each processor with multiple communicating PCS's for a multi-processor system. The clearance of each user partition will be passed on to all MILS PCS devices at start-up by a trusted application security manager. Note that a PCS can be implemented either logically or through hardware in the form of an intelligent I/O device. In [1], we defined a system architecture based on the separation of a MLS system into a MILS system consisting of multiple single level components with a few multi-level components. All components shared a common communication medium. In this earlier work, we used the concept of logical PCS (TNIU's) to mediate communication between separate units.

The PCS's job is to handle enforcement of security access and clearances while CORBA performs the message translation between partitions and processors. As described earlier, some applications will need to accommodate multi-level data while others need to handle only single level data. A similar situation exists in MILS CORBA where some services will be designed to handle multi-level security classifications while other services will remain single level. The current design is for the user parts of the ORB to be loaded with the single level partitions and for multi-level CORBA services to be isolated in its own partition. This will prevent unauthorized access or corruption by user software.

2.3 MILS Application Layer

Applications constitute the third layer in the MILS Architecture and reside on top of the PK and middleware layers. Applications that need to implement specific security policies such as cryptographic or firewall services are assisted via the underlying architecture layers. One of the more important consequences of separating the security mechanisms into layers is that application security can be evaluated separately from other enforcement components. Thus, the application becomes an independent partner in maintaining system-wide security without affecting the other layers [14].

2.4 Advantages of the MILS Architecture

The MILS architecture offers several advantages over past and present high assurance systems [14]. These advantages include:

- Efficient operation within a multi-level environment with support for real-time performance.
- One processor can host multiple applications at different security (or safety) levels.
- Hardware costs are reduced since fewer physically isolated processors are needed.
- Certification costs are reduced since individual functions within non-security critical layers can be certified separately. Non-critical functions can be certified at a lower level.

3. Security Policy

A secure system can be defined as a system that supports a specified security policy (or set of policies). Typically, when a system is designed, there is a single security policy supported by the full system. In the MILS approach, we not only support multiple high-level policies, but we do this through the construction of a system hierarchy, where each layer of the hierarchy supports different security policies, with the lower level policies supporting the higher level policy's implementation.

The most basic aspects of our security policies can be defined in terms of the four provisions:

Data Isolation – Information in the state of one partition must not be accessible to other partitions.

Information Flow Control – This modifies data isolation such that there exists a limited provision for access to other partition's information.

Policy 1.

A single processor system is **Separation Secure** if the following holds:

For all pairs of states of the system, s_1 *and* s_2*,*
For all memory segments of the system, `seg`*,*
where `Policy` *is a function that returns the set of memory segments from which information can flow into the specified segment:*

if
Contents (`Policy(seg)`*) in* s_1 *= Contents (*`Policy(seg)`*) in* s_2 $\wedge$
Current_Partition s_1 *= Current_Partition* s_2 $\wedge$
Contents `seg` *in* s_1 *= Contents* `seg` *in* s_2
then
Contents `seg` *in (top-step* s_1*) = Contents* `seg` *in (top-step* s_2*)*

Where "Contents" determines the data values stores in the specified memory segment(s), and Current_Partition defines the relevant state of the current executing partition. The relevant state includes code, instruction pointer, stack pointers, stack values, and data values.

Figure 4. Single Processor Separation (Policy 1)

Periods processing – Shared resources of the system must be cleansed between context switches. In other words, if there is a memory buffer, register, or other object in the system that is used by one partition, *P*, and then by another partition, *Q*, the data contents of that object from *P* must not be accessible by *Q*. The best method for this is to zero out the contents between uses.

Damage limitation – Faults in a single partition must be contained so as not to affect other partitions. This includes resource exhaustion (such as CPU infinite loops), and memory or code damage. If all interaction between partitions occurs through well-defined boundaries, we can check the interaction at that boundary and respond to faults.

3.1 Single OS Policy

The lowest level of the MILS model provides for a secure, single host operating system – a separation kernel. This kernel must provide for data isolation, information flow, periods processing, and damage limitation. This is the only security policy that the lower level kernel provides. With this policy, we will support the construction of other "higher-level" policies. Consider the separation security policy depicted in Figure 3 (based on the work Greves, Wilding and Vanfleet [7]) as an example of such a policy.

This security policy provides for protection against the two major concerns we have at the lowest level, *infiltration and exfiltration.* The policy states that a "top-step" execution of the system will result in consistent values stored in memory segments. If a system is susceptible to infiltration, then the state of a non-executing partition can affect the execution behavior of the executing partition in an unauthorized manner. By this policy, if `seg` is a memory segment of the current partition's state, the results of the execution are independent of the contents of unauthorized data segments (those not returned by the `Policy` function). In addition, if a system is susceptible to exfiltration, the state of a non-executing partition can be affected by execution behavior of the executing partition in an unauthorized manner. By this policy, if information is not authorized to flow from the current partition to non-executing partition's segment `seg`, the `Policy` function will not list memory segments of the current partition. Therefore, the results stored in `seg` after execution will be independent of the state of the current partition. Infiltration and exfiltration clearly violate the four concepts of the single OS separation security policy.

3.2 Multiple OS Security Policy.

What about multiple-processor systems? Policy 1 uses the concept of information flow between memory segments (including pointers, registers, stacks, etc), but is limited to specifying a single executing partition. When we connect two systems together, through network interfaces, serial ports, Rapid-I/O, we need to address concurrency within our security policy. Again we want a simple policy that can be supportive of

Let OS be the set of operating systems in our composite system.
where `Policy` *is a function that returns the set of memory segments from which information can flow into the specified segment, this includes segments of the current OS:*

For all $os_i \in OS$*, for any pair of states,* s_1 *and* s_2*, of the composite oses*
And for every memory segment, `seg`*, of the composite oses:*

if
Contents (`Policy(seg)`*) in* s_1 *= Contents (*`Policy(seg)`*) in* s_2
Current_Partitions of `seg` *in* s_1 *= Current_Partitions of* `seg` *in* s_2 $\wedge$
Contents `seg` *in* s_1 *= Contents* `seg` *in* s_2
then
Contents `seg` *in (top-step* s_1*) = Contents* `seg` *in (top-step* s_2*)*

Where "Contents" determines the data values stores in the specified memory segment(s), and Current_Partitions defines the relevant state of the current executing partition in each os for partitions whose execution is relevant to `seg`*. In other words, we are only concerned with the execution of partitions in the os local to* `seg`*, and in oses where the current partition, by policy, is authorized to affect* `seg`*. The relevant state includes code, instruction pointer, stack pointers, stack values, and data values.*

Figure 6. Multi-Processor Separation Secure (Policy 2)

higher level applications and policies. Figure 4 depicts a policy that still provides for data isolation, information flow controls, periods processing and damage limitation.

Reading policy 2, there is not much difference between this policy and policy 1. The main difference is that we permit information flow to be specified between the memory segments of different microprocessors. This flow can occur through a network interface, distributed shared memory or other I/O device. We will assume direct transfer using any hardware device. Therefore, our state pairs now consist of a composite state of the states of each OS, and the current partitions of each OS. Infiltration and exfiltration are still a concern, so we must enforce this control throughout the composite system. At first this may appear to be intractable, but it can be done.

Proof:

A proof that a composite system satisfies **policy 2** can be carried out as follows:

Combine the systems such that there exists a well-defined pairing of memory segments in difference oses through which information flows (this will typically be the output and input buffers of the hardware devices)

Prove that each OS satisfies **policy 1** independently

Given the composition nature of **policy 2** we can be guaranteed that the composite system is secure [1].

Further development of the security policies is necessary, along with the formal method templates that can be used by system developers and certification authorities to validate the security claims of the system.

4. Common Criteria Certification

Since the intended use of our architecture is for building high assurance systems, achieving a high level security certification by an accepted evaluation authority is a goal of our design strategy. The current recognized evaluation authority for secure systems in the United States is the National Information Assurance Partnership (NIAPP) group, a joint effort

EAL7 Requirements – Formally Verified Design and Tested

- *Analysis of security functions using a functional and complete interface specification*
- *A high and low-level design of the Target of Evaluation (TOE)*
- *Structured presentation of the implementation to understand security behavior*
- *Formal model of the TOE security policy*
- *Formal presentation of the functional specification and high-level design*
- *Semiformal presentation of the low-level design*
- *Semiformal demonstration of correspondence low-level and high-level designs as appropriate*

Figure 5. Summary of EAL7 Requirements

between the National Security Agency (NSA) and the National Institute for Standards and Technology (NIST). NIAPP uses the Common Criteria (CC) as a certification guideline [9] in the certification of high assurance products. The CC replaced the TCSEC Orange book [6] in the middle 1990's and has undergone several revisions. The current CC version is 2.1 and was published in 1999 [9].

Certification via the CC involves prepackaged levels of assurance, called Evaluation Assurance Levels (EALs). EAL's range from EAL1 to EAL7, which represents the lowest to highest levels of assurance. The EAL's increase in both security and proof requirements with each subsequent EAL level. Up to EAL4 no special security engineering requirements are necessary. Above EAL4, security requirements are more stringent and need to be an integral part of the product. Consequently, for EAL5 and above security functionality cannot be added to an existing product but must be specified along with the non-security requirements. Requiring that security be designed into a product insures that security will be integrated with the product's functionality plus provides certification evidence needed at the highest EAL levels.

The CC is founded on modularity and reuse and supports the idea of a Protection Profile (PP), a security template for a class of products. PP's are generally defined by user communities, developers, or other groups interested in defining a common set of security requirements [9]. Another concept within the CC is that of a Security Target (ST), which is a statement of security claims for a particular security product. A Security Target will often be written for a product that claims to adhere to a particular PP.

The next section details the steps needed to certify a MILS system to an EAL7 level of assurance.

4.1 MILS Certification Steps

The MILS architecture is intended for use at the highest levels of security. Consequently, we are targeting an EAL 7 Assurance Level for the MILS Architecture. CC certification for EAL 7 will include the following steps:

1. Define Protection Profiles for the Partitioning Kernel (PK) [14], MILS CORBA [20] and MILS PCS, which are the main components of the MILS Architecture. It is anticipated that for each component multiple products will be produced that need CC certification. Consequently, product specific ST's will need to be generated from the PK, MILS CORBA and MILS PCS Protection Profiles. Defining PP's for the PK, MILS CORBA, and MILS PCS is a currently an on-going joint effort between interested industry groups, government agencies and several universities [14, 20].
2. Write Security Targets for commercial products that adhere to the PK, MILS CORBA and MILS PCS PP's.
3. Evaluate commercial products that have written ST's for each of the three main components of the MILS Architecture. The ST for a specific product will be needed in the evaluation process to verify that the product has incorporated the security requirements of the target PP. Once the ST has been developed for a MILS product component, an EAL 7 level certification will be performed. EAL 7 is required for these components because they handle multi-level data (i.e. classified, secret, top secret).

   Requirements for EAL 7 are listed in Figure 3. EAL7 requires formal methods models of the security policy and the high level specification, a semi-formal[3] presentation of the correspondence between the high level and low-level design is also required. This will require formal method proofs of the MLS components that are to be evaluated.

   Other components of the MILS Architecture such as system services which provide device drivers and applications do not currently have PP's and will not need to be certified at the highest EAL 7 level. Lower level certification is possible for these other components because of separation and strict communication controls enforced by the MILS Architecture. MLS components can thus co-exist with non-MLS component without compromising overall system security.

5. Other Approaches

The concept of a security kernel for trusted operating systems was first introduced by Schell in 1972 [4]. Since its debut, the security kernel concept has been researched extensively and implemented in several trusted operating systems [13]. Ames et. al. [4] discussed some problems in designing monolithic kernel based systems including:

- Decisions about what functionality should be in a kernel is not simple
- Performance issues often force non-security functions into the kernel
- A constant trade-off between performance, functionality and complexity

Other attempts at security kernel design include KSOS – Kernalized Operating System and SCOMP – Secure Communicating Processes [10]. KSOS is designed hierarchically with a partially ordered set of security partitions, but uses a security kernel and trusted processes to enforce a multi-level security policy [10]. SCOMP provides a security kernel similar to KSOS but achieves its security via hardware through the Honeywell Level 6 machine, which uses a set of rings to enforce security levels.

[3] Semi-formal is defined in the Common Criteria to mean a restricted syntax language with defined semantics (UML is one example).

Another emphasis in secure system design is hierarchical layers for both safety and security. P. Neuman argues that the appropriate use of hierarchical abstraction and encapsulation can produce systems that better satisfy critical requirements for security and safety [11]. Neumann believes that it is possible to divide criticality among different layers in an operating system. He shows how an MLS design can be decomposed so that the lowest layer with the highest criticality is composed of the security kernel followed by the trusted processes, non-security operating system functions, programming languages and user software. Furthermore, he presents risk figures showing that conventional systems have greater potential for disaster from failure or compromise, compared with hierarchically layered systems. He believes that the layered systems can better contain disasters because they are structured. Neuman uses the PSOS –Provably Secure Operating System as an example of a hierarchical system. This system has 16 layers ordered by criticality with lower layers involving hardware, middleware layers relegated to operating system functions and upper layers consisting of user software.

Additional research involving separation and layering was conducted by Rushby and is the most similar to the MILS architecture. As stated in Section 1, Rushby is credited with defining and proving separation properties for security [15, 17]. In [15] Rushby discussed problems with the security kernel imposing a single security policy over the entire system. Logical separation of security functionality allows easier verification of security since components are separated in self-contained partitions. This view of separation has been incorporated into the MILS Architecture. A later study by Rushby [18] applied separation concepts to embedded systems; a layered architecture was proposed based on a Domain Separation Mechanism, resource manager and application manager. Again, the separation of security functions into layers with differing security policies is similar in concept to the MILS Architecture. Another paper by Rushby and Randell [19] showed that multi-level security could be implemented in a distributed system. The authors proposed that key concepts such as separation and mediation should be distinguished and implemented by separate mechanisms. They proposed constructing a secure distributed system via mediation devices acting as individual reference monitors.

6. Conclusion and Future Work

In this paper we have presented a multi-layer architecture, the MILS architecture, that supports multiple independent levels of security. The targeted environment for this architecture is the embedded, real-time world of avionics and military communication systems. We emphasize that MILS is not intended for general purpose computing.

The concept of separation is key to the success of the MILS Architecture. Separation has been used in constructing security policies at each architecture layer, in the mechanisms that enforce the security policies, and in the certification of the high assurance components.

The MILS Architecture represents the culmination of research ideas that have been discussed in the security literature for the past 20 years. Development of MILS is actively supported by government, industry, and university groups. Ultimately, the goal is to produce a viable MILS implementation that can be used by government and industry developers for high assurance, multi-level systems.

Our specific role in the MILS architecture development process is to assist in the MILS CORBA effort. Thus, the future research discussed here relates specifically to MILS CORBA as opposed to the entire MILS Architecture. These next steps include:

- Finishing the MILS CORBA Protection Profile
- Obtain NIAP approval for the MILS CORBA Protection Profile
- Develop abstract formal methods artifact templates from MILS CORBA specifications (to be used in later EAL7 certification efforts)
- Develop formal methods proof sketches (to be used during the implementation of MILS CORBA to produce formal methods artifacts for EAL7 certification)

Similar additional steps will need to be taken by other groups in achieving certification for the Partitioning Kernel and other multi-level architectural components.

7. References

[1] Alves-Foss, J. "The architecture of secure systems." In Proc. Hawaii International Conference on System Sciences, Vol. III, pp. 307-316, Jan. 1998.

[2] Ames, B. "Real-Time Software Goes Modular", Military & Aerospace Electronics, Vol 14, No. 9, pp24-29, Sept. 2003.

[3] ARINC, "Avionics Application Software Standard Interface", ARINC Specification 653, Aeronautical Radio, Inc., Annapolis, MD, Jan 1997.

[4] Ames, S. R., M. Gasser, R. R. Schell. "Security kernel design and implementation:

An introduction." IEEE Computer, Vol. 16(7), pp. 14-23, Jul. 1983.
[5] Bolton, F. Pure CORBA, Sams Publishing, 2002.
[6] DOD. Trusted Computer System Evaluation Criteria, DOD5200.28-STD, Dec. 1985.
[7] Greve, D. and M. Wilding and W.M. Vanfleet. "A Separation Kernel Formal Security Policy", In Proc. ACL2 Workshop 2003, July 2003.
[8] Keahey, K. A brief tutorial on CORBA, http://www.cs.indiana.edu/~kksiazek/tuto.html.
[9] NIST. Common Criteria for Information Security Evaluation, available: http://csrc.nist.gov/cc/ccv20/ccv2list.htm, NIST, 1999.
[10] Neumann, P. G. "Experiences with formality in software development," In Theory and Practice of Software Technology, D. Ferrari, M. Bolognanai, and J. Goguen (eds.), pp. 203-219, 1983.
[11] Neumann, P. G. "On hierarchical design of computer systems for critical applications," IEEE Transactions on Software Engineering, Vol. SF-12(9), Sept. 1986.
[12] OMG, CORBA Security Services Specification, Vol. 1.7(draft), 1999.
[13] Pfleeger, C. P. Security in Computing, Prentice Hall, Upper Saddle River, NJ, 1997.
[14] Rockwell Collins, Boeing, et. al. Partitioning Kernel Protection Profile, Ver. 1.02(draft), Feb. 2003.
[15] Rushby, J. "Design and verification of secure systems," In Proc. ACM Symposium on Operating System Principles, Vol. 15, pp. 12-21, 1981.
[16] Rushby, J. "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance." Technical Report, Computer Science Laboratory, SRI International, March 1999.
[17] Rushby, J. "Proof of separability: A verification technique for a class of security kernels," In Proc. International Symposium on Programming, Lecture Notes in Computer Science, Vol. 137, pp. 352-367, 1982.
[18] Rushby, J. "A trusted computing base for embedded systems," In Proc. 7th DOD/NBS Computer Security Conference, (Gaithersburg, MD, Sept. 24-26) pp. 294-311, 1984.
[19] Rushby, J. and B. Randell. "A distributed secure system," IEEE Computer, Vol. 16(7), pp. 55-67, Jul 1983.
[20] University of Idaho, MILS CORBA Protection Profile, Ver. 0.52(draft), Aug 2003.
[21] White, P., M. Van Fleet, and C. Dailey. High assurance architecture via separation kernel, internal communications, 1996.