The MILS Architecture for High-Assurance Embedded Systems

Jim Alves-Foss, W. Scott Harrison, Paul Oman and Carol Taylor

***Abstract*— High-assurance systems require a level of rigor, in both design and analysis, not typical of conventional systems. This paper provides an overview of the Multiple Independent Levels of Security and Safety (MILS) approach to high-assurance system design for security and safety critical embedded systems. MILS enables the development of a system using manageable units, each of which can be analyzed separately, avoiding costly analysis required of more conventional designs. MILS is particularly well suited to embedded systems that must provide guaranteed safety or security properties.**

***Index Terms*— Multi-level Secure, High-Assurance, MILS**

I. Introduction

The design and implementation of secure applications is a daunting task. Especially since these applications rely upon the security of the underlying operating system and services. Too often, a secure service or application will be compromised by a security flaw in a supporting service or operating system. The *Multiple Independent Levels of Security and Safety (MILS)* approach remedies this situation by providing a reusable formal framework for high assurance system specification and verification.

High assurance systems are those that require convincing evidence that the system adequately addresses critical properties such as security and safety [1]. If the high assurance system fails to meet its critical requirements, then there is a potential for security breach or loss of life. Since such systems are so critical, there is a need for a rigorous design and analysis process. The avionics community has understood this for years and has developed a set of guidelines for the design, analysis and evaluation of safety systems [2], [3]. Although very rigorous and adequate for safety of airborne computing systems, it is insufficient to meet the security concerns of high-assurance security systems; systems that protect national security interests. The Common Criteria (CC) provides guidance for design, analysis and evaluation of security critical systems that includes a very high level of assurance [4]. At the higher levels of assurance, the CC requires the use of formal methods, mathematical models and proofs.

It is commonly believed that the successful deployment of a high assurance secure application requires the existence of an underlying secure operating system and services. Unfortunately, attempts to develop secure, general-purpose operating systems have failed to be supportable. They have failed for various reasons, but predominately due to the fact that they were architected to do too much and the requisite formal methods used became too difficult to manage. Historically, a high assurance operating system was based on the concept of a security kernel and Trusted Computing Base (TCB), all of which required formal methods evaluation. Two such systems that became unsupportable are Blacker [5] and Caneware [6]. Through the development lifecycle they included more and more functionality into the trusted computing base, requiring too much effort in the development of the supporting formal methods.

For embedded systems, the need for a secure operating system is no less critical. While user access is not a problem, secure communications and safe process execution is still a concern. Careful development and verification of the operating system and trusted applications must assure that the system is free from security vulnerabilities.

To obtain high assurance we recommend a hierarchical system architecture, where multiple layers provide specific, well-defined security mechanisms that can be used by higher layers. When a system is designed to provide a security mechanism, the mechanisms must be i) always invoked, ii) non-bypassable, iii) tamperproof and iv) evaluatable. Evaluation of the correctness of the mechanisms is a time consuming and difficult task, made even more difficult by complex security mechanisms.

Fortunately, a sound engineering approach (the MILS approach) exists to simplify the specification, design and analysis of security mechanisms. This approach is based on the concept of separation, as introduced by Rushby [7], [8]. The concept of separation has been accepted in the avionics community and is a requirement of ARINC 653 [9] compliant systems. Through separation, we can develop a hierarchy of security services, where each level uses the security services of a lower level to provide a new security functionality that then can be used by higher levels. 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. General user applications can then execute, untrusted, isolated from each

Manuscript received August 18, 2004, revised December 15, 2004. This material is based on research sponsored by AFRL and DARPA under agreement number F30602-02-1-0178. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL and DARPA or the U.S. Government.

All authors are with the Center for Secure and Dependable Systems at the University of Idaho, Moscow ID 83844. E-mail: [jimaf,ctaylor]@csds.uidaho.edu; [harrison,oman]@cs.uidaho.edu. Website: http://www.csds.uidaho.edu

other except through communication channels managed by these security services.

In the remainder of this paper we provide an overview of the MILS architecture and its associated levels. In Section II, we introduce the MILS architecture, the system design model that is the focus of our work. In Section III, we present the partitioning kernel, the lowest layer of the MILS hierarchy. In Section IV, we discuss the hardware support needed to a separation kernel. Then in Section V, we discuss the needs of shared device drivers. Section VI expands on the basic MILS architecture for distributed systems. This is complemented by the discussion of MILS middleware and applications in Section VII and Section VIII. Sections IX, X, and XI conclude the paper with a description of an example MILS system.

II. MILS Architecture

In the past, secure systems were designed with the concept of a security kernel and a Trusted Computing Base (TCB) [10]. The key concept behind this approach is that the security decisions and the security enforcement mechanisms are an integral part of the TCB. Following this design paradigm, development teams found that more and more of their system functionality was being included in the TCB. Once this occurred, the evaluation of system security became unmanageable.

What is needed is a system architecture that allows a structured, compartmentalized approach to the design of a secure system. As we shall see, this design necessarily requires the deployment of several different execution environments (partitions/tasks) within a microprocessor. This design feature will force the system kernel to support many context switches per second – on the order of thousands. Fortunately, we are now at a point where the speed of current processors will support this level of context switches for the type of lightweight system kernel we propose. Evidence of progress in this direction can be seen in recent development efforts such as the ARINC 653 Standard [9] for partitioning operating systems, the Motorola AIM/Mask [11] separation-based operating system and encryption chipset, and the Integrity-178 [12] partitioning Real-Time Operating System.

The focus of this research is based on the concept of the MILS architecture, which was created to simplify the process of the specification, design and analysis of high-assurance computing systems [13].

Within the MILS architecture, application layer entities are provided with the mechanisms to control, manage and enforce their own application level security policies in a manner that ensures that the enforcement mechanisms are always invoked, non-bypassable, tamperproof and evaluatable. At this level we have trusted application-level security services and untrusted applications. The MILS architecture assumes certifiable trust within the microprocessor, separation kernel, middleware services layer, and for the application-level security services. Thus, we assume a benign fault model within the microprocessor and RTOS, but a malicious fault model for the untrusted applications. Benign faults will need to be addressed via traditional fault-tolerant diversity and redundancy, with fault containment procedures instituted within the microprocessor and RTOS. For example, illegal instructions and memory violations are assumed to be trapped and handled via the microprocessor and separation kernel. Likewise for covert channels and residual data needing sanitization. Violations of information flow that cannot be detected and trapped in this fashion will need to be handled within the middleware services layer. Faults occurring within the middleware services layer (those that cannot be detected and trapped at lower levels) constitute an open issue currently being addressed by the MILS research community. At the application layer, execution is confined to the application partition, with limited communication to other partitions. All communication is monitored by the certified application layer security services and lower processing levels. Applications are assumed to be rogue and are confined to operating within their partition resources, with finite boundaries of space and time. In the remainder of this section we introduce the concepts behind the MILS architecture.

A. MILS, MLS, MSLS and SLS

Traditionally, the military model of a secure operating system includes the concept of multi-level security (MLS). The idea behind this concept is that the system will be processing data items that are classified at different levels of security, and the information flow security policy that prevents the transfer of high-level classified information into low-level objects must be preserved. Therefore, we define a MLS system as one that must be certified to process and output co-mingled data at multiple classification levels. Classic security models, such as the Bell-LaPadula model [14], have been used to specify the secure behavior of such MLS systems.

The problem with full MLS systems is that they must be rigorously analyzed for security before they can be certified. Every portion of the MLS system must be analyzed to ensure that it properly handles labelled data and that there is no possible violation of the security policy. Even with a TCB architecture, or reference monitor [15], in place, there is often too much to evaluate.

The MILS architecture was developed to resolve the difficulty of certification of MLS systems, by separating out the security mechanisms and concerns into manageable components. These components are classified based on the way they process data:

- **SLS** Single-Level Secure component that only processes data at one security level.
- **MSLS** Multiple Single-Level Secure component that processes data at multiple security levels, but always maintains separations between classes of data. A device that processes messages one at a time (such as an I/O device driver) may be such a device.
- **MLS** Multi-level Secure components that co-mingle data at different security levels. Typically this is a device that will downgrade information from a higher level of security to a lower level through either filtering or the application of encryption technology.

A MILS system isolates processes into partitions, which define a collection of data objects, code and system resources.

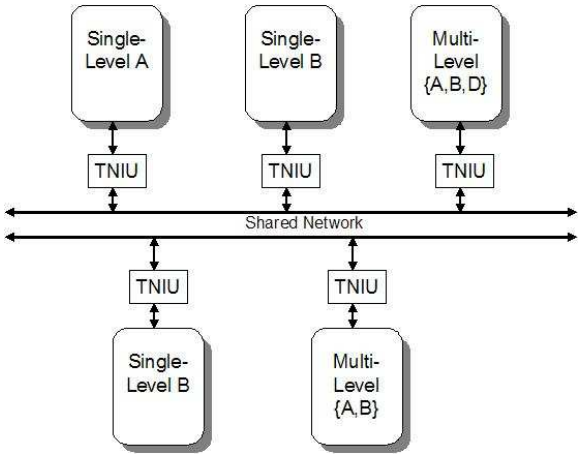

Fig. 1. MILS refinement of an MLS system

These individual partitions can be evaluated separately, if the MILS architecture is implemented correctly. This divide and conquer approach will exponentially reduce the proof effort for secure systems. To support these partitions the MILS architecture is divided into layers.

B. System Architecture View

Alves-Foss [16] defined a system architecture based on the separation of an 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, (see Figure 1). In this work the author used the concept of logical trusted network interface units (TNIU) [17], to mediate communication between separate units. As mentioned in that paper, this mediation can be implemented by the operating system, in this case a MILS separation kernel (SK), which limits communication between partitions to only that which is specifically configured into the system. The system architect, using an SK, can graphically represent the system and authorized information flow.

The architecture in Figure 1 can represent a collection of SLS, MSLS and MLS components in a system, for example a collection of microprocessors or computers on a shared network. If the communication to the network is mediated by TNIUs, we get the configuration defined by Rushby and Randell [17], which was formally specified and verified to satisfy the restrictiveness [18], [19] security policy in [20], [21].

However, we are not limited to a physical network or bus for communication. If we virtualize this network into representing kernel supported communication channels, then we get the type of separation system required by the MILS architecture, as specified in [16]. For example, we can take a multi-level file server and architect it as depicted in Figure 2. Regardless of the view, the architecture can refine MLS components down to a network of single level components, MSLS components, and simpler MLS components. With the SK and middleware in place, the architect can be confident of system security and safety. In addition, the certification of the components and layers is modular, allowing for great reuse.

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 SK-supported data streams. This view of the system can be depicted more abstractly as a directed graph with vertices for the partitions and edges for the channels; instead of the broadcast network show in Figures 1 and 2.

III. MILS Separation Kernel (SK)

The partitioner (separation kernel) layer is the base layer of the system, and is responsible for enforcing data separation and information flow controls within a single microprocessor; providing both time and space partitioning. This layer provides only a few base security mechanisms, following the recommendations of Saltzer and Schroeder [22] for economy of mechanism, keeping security mechanisms as simple as possible. The complexity of the partitioner is low enough that it can even be implemented in the microcode of a partitioning microprocessor as shown in [23]. The partitioner provides for the following:

1) *Data Separation.* The memory address spaces, or objects, of a partition must be completely independent of other partitions. The act of accessing an object by an executing partition must not affect the state of other partitions (no *exfiltration*), and an executing partition must not be affected by the state of any other partition or partition's objects (no *infiltration*).
2) *Information Flow.* This requirement is a modification of data separation. Although pure data separation would be easier to verify [8], it is not practical. There is a definite need for partitions to communicate with each other. However, for secure systems, we need to be able to define the authorized communication channels between partitions. The SK will define precise moderated mechanisms for inter-partition communication. Only through these mechanisms may pure data separation be violated.
3) *Sanitization.* To ensure the information flow requirement, the SK is responsible for cleaning any shared resources (microprocessor registers, system buffers, etc.) before a process in a new partition can use them.
4) *Damage Limitation.* The consequences of a fault or security breach in one partition are limited by the data separation mechanisms. Addresses spaces of partitions are separate, and as such, an errant process in one partition can not affect processes in other partitions. The SK will also enforce bounds on shared resource, providing guaranteed minimum processing time, memory and other resources to the partitions as well as enforcing maximum usage of these resources.

In addition, the partitioner must be always invoked, non-bypassable, tamperproof and evaluatable. In the MILS architecture, the partitioner is responsible for timesharing the microprocessor between the partitions, and this function cannot be stopped. For high assurance systems satisfying the EAL7 certification requirements of the Common Criteria (CC), the

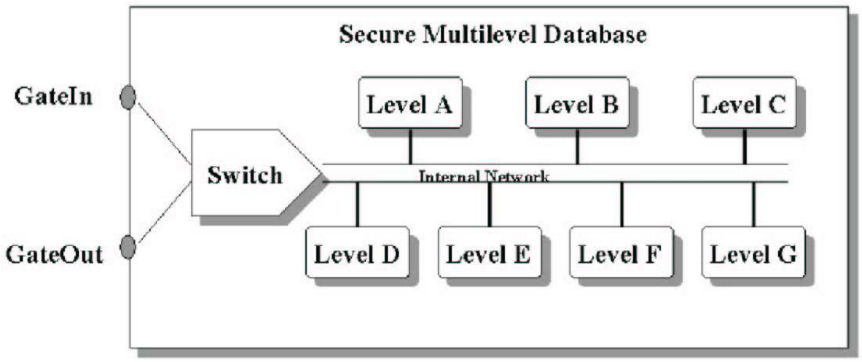

Fig. 2. Whitebox Representation of Secure MultiLevel FileServer DataBase

functionality of the partitioner must be certified to have been rigourously verified using formal methods [4]. A CC protection profile has been developed to define the functionality of such a high-assurance partitioner [24], and submitted to the Open Group. One example of a partitioner specification can be found in ARINC 653 [9].

IV. MILS Hardware Support

A system built on the concept of separation requires a certain amount of hardware support for the SK to work correctly. According to the Separation Kernel Protection Profile (SKPP) [24], and ARINC-653 [9] the hardware support for a SK includes:

- **Processing Power.** For any system, but especially real-time systems, the processor must have sufficient computing capacity to meet the worst-case timing requirements of the system.
- **Atomicity.** The processor must provide atomic operations for implementing processing control constructs, such as partition swaps and memory map changes.
- **Privileged mode of operation.** There will be some privileged instructions that must only be executed by the SK.
- **Memory Management Unit (MMU):** The MMU provides separation of address spaces between the partitions. Without hardware support for separation, there can be no data isolation or damage limitation. The processor must have access to the required memory resources and provide the SK with the ability to restrict partition access to memory.
- **Instruction Traps**. The processor must have some mechanism to transfer control to the SK if a partition attempts to execute a privileged or invalid operation.
- **Timing Control.** The processor must provide the SK with the ability to control and restrict the execution time of partitions; and therefore must have access to timing resources which provide a non-bypassable way of ensuring control is returned to the SK after some elapse of time.
- **I/O Access Limitation.** The processor must provide a mechanism for restricting access to I/O devices to specific partitions; and therefore have access to required I/O resources.

This basic list of processor features is available on many commercial microprocessors and motherboards; and should not be seen as a hinderance in the development of a secure MILS systems. The hardware used in MILS systems may be little different than the hardware currently used in embedded systems and can include commercial off-the-shelf hardware when available. For high assurance certification there may be requirements that the motherboards be certified, however this is not a specific requirement of the MILS architecture.

V. MILS Device Drivers

The MILS architecture requires that the SK remains small, to ensure that it can be fully evaluated. This means that services typically included in the operating system must now be included in the address space of the individual partitions, or delegated to separate “shared” partitions, with communications between partitions mediated by the SK policy. Device drivers for shared devices fall into this later category. Devices in a MILS system could include sensors, controllers and possibly mass storage.

When a device is private and should not be shared, it can be assigned to its own partition. Since most modern processors use memory-mapped I/O, the device can be protected from access by the MMU. An example of a critical device driver that should not be shared is the controller for the landing gear in an aircraft. Other less sensitive devices could be shared between several partitions. A sensor that places a sample in a device register is an example of a device that might be accessible to more than one partition. An engine temperature sensor would periodically update a data register with the latest reading and would need to be available to the process that updates the pilot’s display as well as the engine overload warning system.

Devices in a high assurance system can also be considered critical and in need of privacy for security reasons. Intelligence agencies require that data at different classification levels not be stored on the same disk. Yet there is often a need to handle information from several classification levels. An MLS file system would need to separate data from partitions running at different classification levels and represents a shared storage device. The MLS file system would be isolated in its own partition with carefully defined communication paths between partitions at different classification levels. An example of a need for a shared storage device would be a military aircraft

that must juggle data from radar, targeting activities and communication with traffic control.

VI. Distributed MILS

To improve performance, individual partitions within enclaves can be mapped to separate processors (see Figure 3). An enclave is a group of partitions that are running at the same classification level. To provide MILS separation, this requires support at many levels. Of the four requirements of MILS systems, data separation, damage limitation and sanitization are no longer of concern between partitions on separate processors since there are no shared resources between these partitions. However, information flow controls must still be enforced.

Support for a distributed MILS system requires that the communication between processors be managed by the MILS system. If the inter-processor communication medium is open (i.e., accessible by non MILS managed components), then the communication must be cryptographically protected. However, if the medium is closed, then it may be possible to avoid the expense of encryption. If all processors run a MILS separation kernel, then the kernel can enforce network communication through an appropriate trusted device driver, as discussed in the previous section. If some processors in the system are running untrusted operating systems, we can use the TNIU approach, as discussed in Section II.

Therefore, whether we have open or closed networks, all MILS components, or a mix of MILS and non-MILS components, we can architect the system to enforce the information flow requirements of the MILS system.

VII. Middleware Services Layer

The middleware services layer provides for an extended scope of the separation concepts introduced by the partitioner. These services are dependent upon the needs of the specific application and are not constrained by the MILS architecture. They can include services such as resource allocation of shared data storage devices, object-oriented inter-partition communication, communication services between partitions on multiple processors, or real-time data distribution services. Middleware services are concerned about end-to-end data processing, and not just the single microprocessor data processing of the partitioner. 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 labelling 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) *Labelling.* The middleware layer must ensure that messages sent between individual partitions are correctly labelled with the sender's security classification and unique identity.
2) *Filtering.* The middleware layer is responsible for filtering out any messages that are not appropriately labelled 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. This is true even if the two partitions are labelled with different security classifications. Middleware services provide secure information flows between partitions, typically via protocol-specific labelling and filtering. For example, a GIOP-guard would ensure that CORBA GIOP messages are formatted and routed correctly.

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. If all partitions within an enclave are labelled with the same security classification, then we have a secure system as depicted in Figure 4.

Each partition of this configuration processes information at the security level of its enclave, and is called a *single level secure* (SLS) partition. This differs from a high-water mark system, which is a single level system in which all partitions process data at the same security classification, a configuration we wish to avoid, because they would all need to process the data at the highest level for it to be secure. If we wish to allow communication between enclaves, we can graphically specify that as in Figure 5, which will be discussed in more detail later.

VIII. Application Layer

The application layer is responsible for enforcing application layer security policies. Traditionally, MLS has been defined as data from more than one security classification. In MILS this definition has been refined into two distinct parts: multi-level secure (MLS) and Multi-Single Level Secure (MSLS).

An MLS component in a MILS system is one that deals with multiple classifications and transforms the data from one classification level to another. Because of the potential seriousness of violating its security policy, MLS components require the highest level of scrutiny and verification.

MSLS processes or devices also handle multiple data classifications but separate the data into independent streams with no communication between streams. Consequently, there is little danger that highly classified data will flow to unclassified entities assuming the MSLS device is functioning correctly. MSLS devices also need to be carefully verified that they maintain a separation secure environment, but they need a lower level of verification effort than the MLS components. Examples of these component types include the following:

1) *Collator.* A collator receives data from multiple classification levels, processes that data and transmits data at a single higher classification level. A collator is an

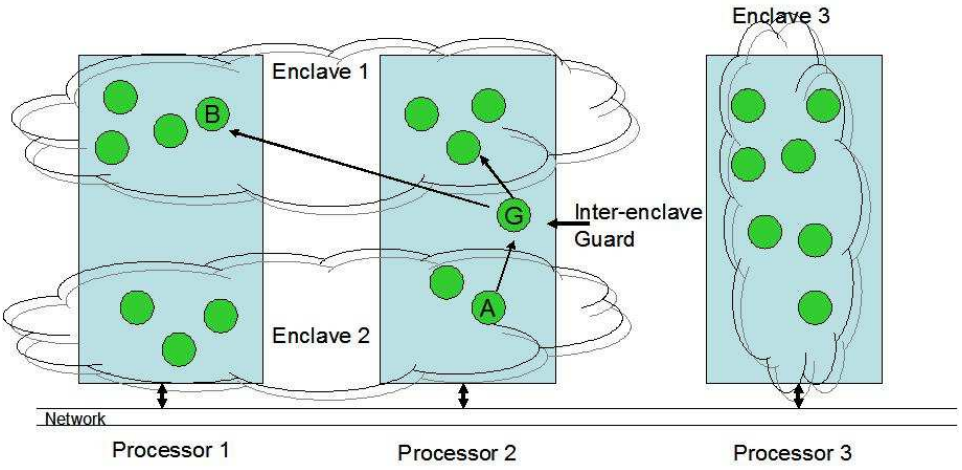

Fig. 3. Enclaves distributed over separate processors.

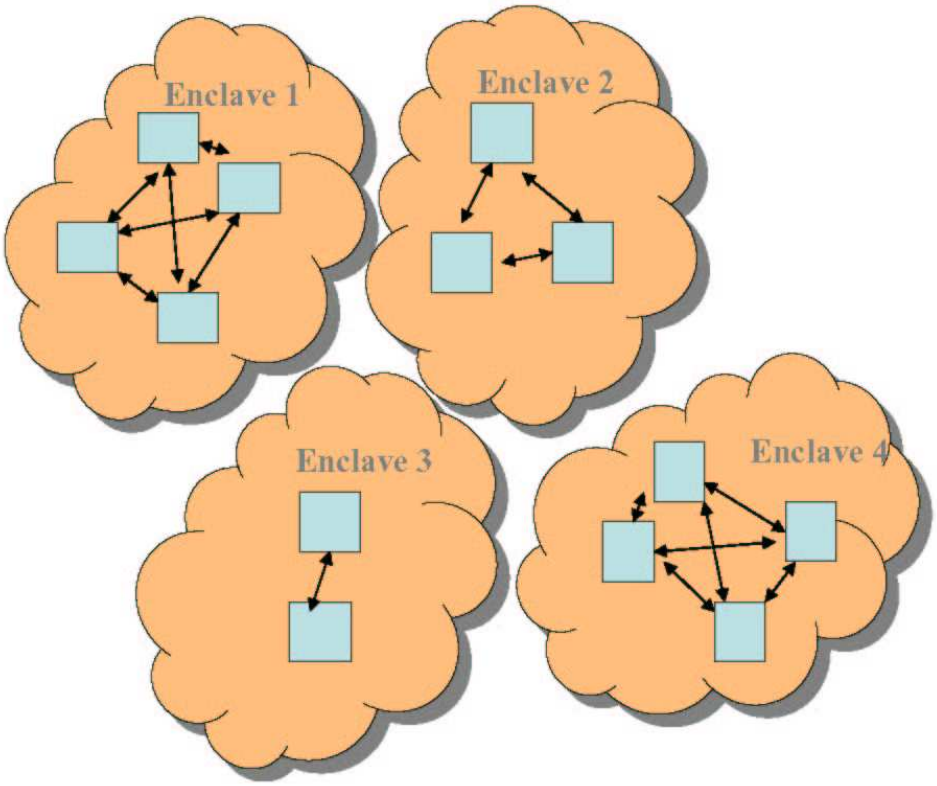

Fig. 4. System of 4 independent secure enclaves

example of a MLS device. This type of system is secure if the middleware layer can ensure that the transmission of information to the collator is one-way, that there is no feedback or response mechanism to signal the sender. If the middleware can be designed to truly support one-way communication, possible covert channels are automatically removed and this type of component is guaranteed to be secure.

2) *Downgrader.* A downgrader transmits data at a security classification level that does not dominate the highest level of input it receives. A downgrader, by definition, is necessarily an MLS component and must be independently evaluated for security. However, in the MILS architecture, the evaluation of the downgrader can be limited to processing within the downgrader's partition. We are already guaranteed data isolation and information flow control by the MILS architecture, and can take these properties as axioms for the higher layers. This greatly reduces system verification efforts. Further reduction results from careful system design as discussed in Section II-B when referring to the secure multi-level file system/database.
3) *Encrypter.* An encryption device is one where a plain text data stream is transformed into a non-readable

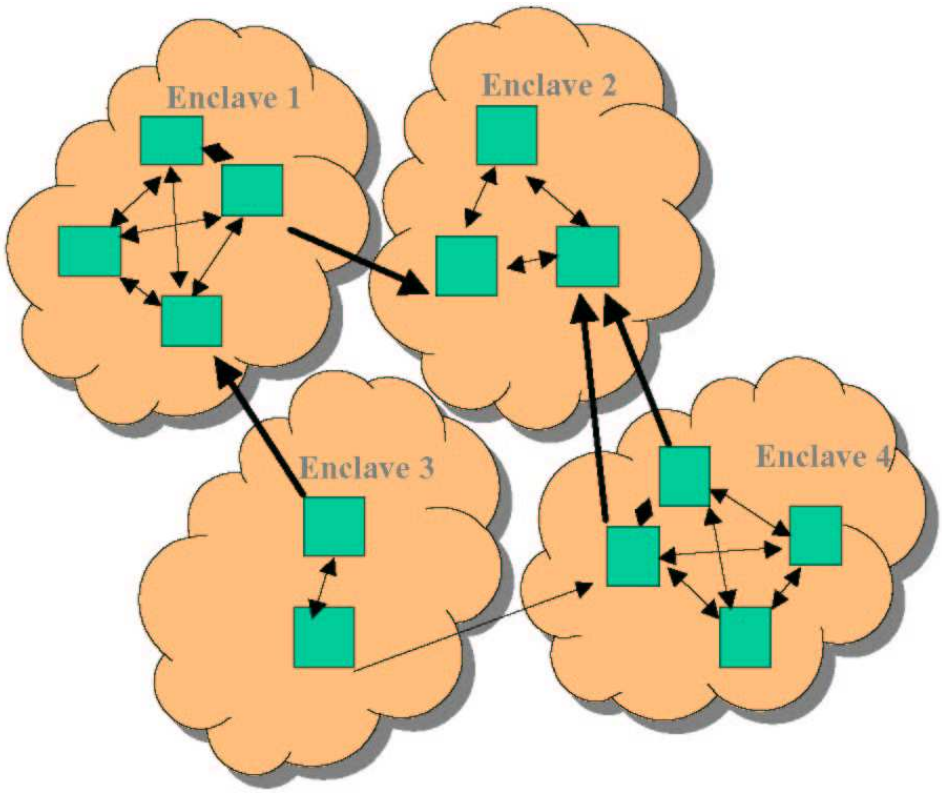

Fig. 5. System of communicating independent secure enclaves

encrypted stream according to a mathematical algorithm. Typically, data at a higher classification level is encrypted and sent out over a lower classified device. Thus, an encrypter is by definition an MLS device.

4) *MILS Message Router (MMR).* A MILS Message Router will function as a data switch by taking data from multiple partitions at various classification levels and routing the messages to the correct destination, which may include additional trusted devices that determine if the message satisfies the application-level security policy. Messages will be checked to insure authorization exists for communication between the partitions. Since transformation between classification levels is not being performed, the MMR is an example of a MSLS device.

The ability to isolate the portions of the system that necessarily process MLS data enables us to focus our resources and limit our verification efforts, making high assurance components realizable. The system architect effectively builds into the system a set of *software firewalls* that are responsible for implementing the application layer security policy with confidence that the middleware and SK will ensure that the application layer mechanisms are non-bypassable, tamperproof, always invoked, and evaluatable.

The MILS architecture now permits us to reuse the layers. We can port a middleware package from one certified SK to another with greatly reduced additional proof effort. We can avoid the process where each verification effort restarts from scratch; instead we reuse the certified layers in the architecture.

IX. A Secure System Using MILS

Consider again the system depicted in Figure 3. If partition A in *Enclave 2* wishes to send a message to partition B in *Enclave 1*, we need to be sure that the message sent does not violate the security policy. There are two scenarios we need to investigate:

1) $A \leq B$. In this configuration, the security level of A is less than that of B. In other words, information is permitted to flow freely from A to B. Since the MILS system enforces information flow policies dictated by the system configuration, this flow is controlled by the SK and the Distributed MILS system.
2) $A \not\leq B$. In this configuration, the security level of A is not less that the security level of B. Now, information is not permitted to flow freely from A to B. Given this restriction we have two choices:
   - The first is that A has been evaluated and certified to be *trustworthy*. This means that it has been shown that any information A does send to B is not in violation of the security policy. For example, A could be a cryptographic engine which is certified to encrypt any messages it sends to B.
   - The second is when A is not trustworthy. In this case we need a trusted intermediary (a guard) to control information flow from A to B. This guard is a trustworthy application which is responsible for analyzing the content of the communication and determining whether this communication is in accordance with the system security policy. The

guard has the ability to modify the contents of the message, delete the message or send a constructed response back through the MMR.

X. Example MILS System

We are not limited to having each node of the graphs of Figures 4 and 5 represent a partition. For example, in a shared-memory system, the system can be viewed as the directed graph depicted in Figure 6, where partitions are only connected to memory segments and memory segments are only connected to partitions. Interprocess communication is implemented through reading and writing of shared memory buffers[1]. Partitions can read or write a connected memory segment depending on the direction of the edges.

In this example, the system depicted is a secure crypto communication device. This device receives data from the *red* network on the left, through the network interface (RPM). This data is assumed to be correctly labelled. Based on the labelling the system sends the data to one of three encryption engines (Type 1 devices), each implementing a single encryption algorithm, and maintaining separate encryption keys. The encrypted messages are then transmitted on the insecure *black network*. Incoming encrypted messages from the black network are appropriately sent to the correct decryption engine. The results are then labelled correctly and sent out over the red network.

The MILS system depicted in this figure forces isolation between partitions. The only authorized information flow occurs through the shared data segments as depicted in the graph. Input into the Red Switch (RS) is separated multi-level data, therefore the RS device is an MSLS application that enforces the system security policy by correctly transmitting data to the appropriate shared memory buffer for the encryption engines. Similarly the Red Verifier (RV) receives single level information from the decryption engines and combines it to transmit out on the red network (appropriately labelled). This device is also necessarily an MSLS device. Given the existence of a verified MILS SK, the only two components which need verification are RS and RV (possibly RPM depending on how security labels are managed). Every other component is single level, or is a separately verified Type-1 encryption device, and does not need security verification. This reduction in verification effort was capitalized on by the AIM system developed by Motorola, which uses the MASK separation kernel [11].

XI. Conclusion

High assurance systems, whether they be security critical or safety critical, require an extensive amount of analysis. In this paper we have discussed the MILS architecture, a design approach for high assurance systems that enables manageable analysis through modular design and layered enforcement of security policies. At the lowest level of enforcement, the MILS architecture supports the policy of data isolation, information flow, damage limitation and sanitization. Specific higher level security policies must be enforced by trusted partitions utilizing the resources of lower level policies. The lower levels support the enforcements mechanisms of the higher levels, working as partners to support the application level security policy.

The MILS architecture is not just an academic exercise, but rather an approach to system design that is supported by industry and government. Partitioning kernels, the lowest layer of the MILS architecture are already being deployed by multiple real-time operating system vendors [12]. Common criteria protection profiles are currently being developed for both the partitioning kernel [24] and MILS middleware. At the University of Idaho, we are currently developing a testbed for MILS concepts, which had been built, in its entirety, from COTS components. The testbed consists of four single board computers and the Integrity and LynxOS-178 RTOS operating systems.

In addition, there is much work currently being done on the necessary formal methods for efficient mathematical modelling and analysis of high assurance systems. We have already developed proofs for the separation maintained by an exemplary separation-based microprocessor [23]. Greve et al. developed a security policy for separation kernels and have shown how it can be used in a two-level hierarchy [25]. We have evaluated this work and discussed the uses and limitations of it [26]. Results from our testbed and formal proofs will be forthcoming in publications from the authors.

[1]In general, MILS inter-partition communication may occur through any verified communication channel offered through the kernel.

References

[1] M. P. Heimdahl and C. L. Heitmeyer, "Formal methods for developing high assurance computer systems: Working group report." in *Proceedings, Second IEEE Workshop on Industrial-Strength Formal Techniques (WIFT'98)*, Oct 1998.

[2] *Software Considerations in Airborne Systems and Equipment Certification (RTCA DO-178b)*, RTCA Std., Dec 1992.

[3] *Requirements Specification for Avionics Computer Resource (ACR) (RTCA DO-255)*, RTCA Std., June 2000.

[4] *Common Criteria for Information Technology Security Evaluation, Version 2.1*, Common Criteria Project Sponsoring Organization Std., August 1999.

[5] C. Weissman, "BLACKER: Security for the DDN examples of A1 security engineering trades," in *Proc. IEEE Symposium on Research in Security and Privacy*, 1992, pp. 286–292.

[6] H. Rogers, "An overview of the CANEWARE program," in *Proc. 10th NIST-NCSC National Computer Security Conference*, 1987, pp. 172–174.

[7] J. Rushby, "Design and verification of secure systems," in *Proc. ACM Symposium on Operating System Principles*, vol. 15, 1981, pp. 12–21.

[8] ——, "Proof of separability: A verification technique for a class of security kernels," *Proc. International Symposium on Programming, Lecture Notes in Computer Science*, vol. 137, pp. 352–367, 1982.

[9] *Avionic Application Software Standard Interface (Draft 3 of Supplement 1) (Specification ARINC 653)*, ARINC Std., 2003.

[10] *Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense Computer Security Center Std. DoD 5200.28-STD, December 1985.

[11] W. Martin, P. White, F. Taylor, and A. Goldberg, "Formal construction of the mathematicaly analyzed separation kernel," in *Proc. of the* 15^{th} *International Conference on Automated Software Engineering*, 2001, pp. 133–141.

[12] B. Ames, "Real-time software goes modular," *Military & Aerospace Electronics*, vol. 14, no. 9, Sept. 2003.

[13] P. White, W. Van Fleet, and C. Dailey, "High assurance architecture via separation kernel," October 2000, draft.

[14] D. E. Bell and L. J. LaPadula, "Secure computer systems: Unified exposition and multics interpretation," The MITRE Corporation, Bedford, MA, Tech. Rep. MTR-2997, July 1975.

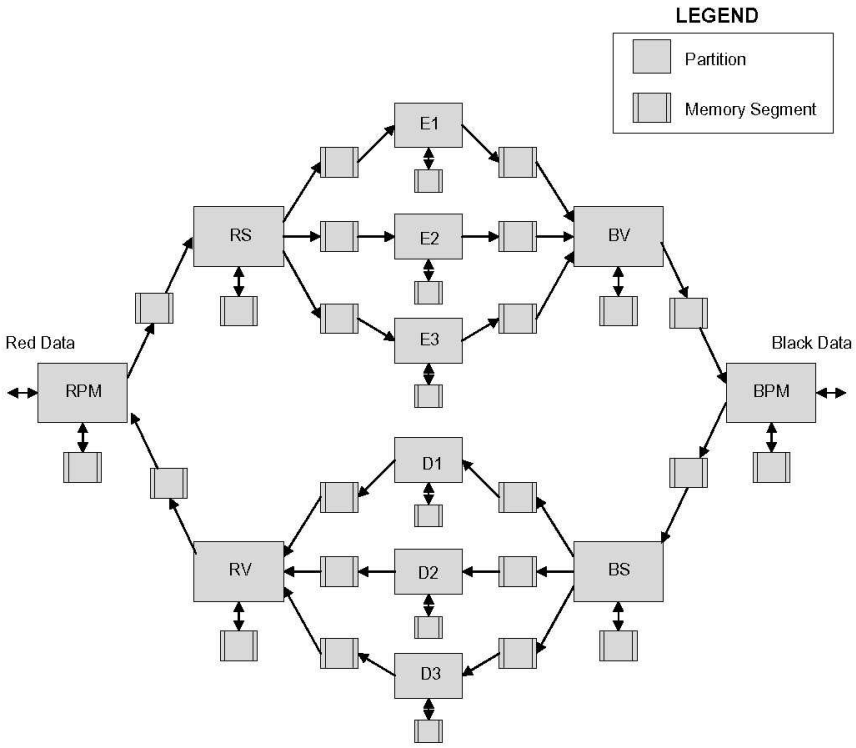

Fig. 6. System Graph for Shared Memory Architecture

[15] J. Anderson, "Computer security technology planning study," USAF Electronic Systems Div., Bedford, Mass., Tech. Rep. ESD-TR-73-51, October 1972.
[16] J. Alves-Foss, "The architecture of secure systems," in *Hawa'ii International Conference on System Sciences*, Jan. 1998, pp. 307–316.
[17] J. Rushby and B. Randell, "A distributed secure system," *IEEE Computer*, vol. 16, no. 7, pp. 55–67, 1983.
[18] D. McCullough, "Noninterference and the composability of security properties," in *Proc. IEEE Symposium on Security and Privacy*, 1988, pp. 177–187.
[19] ——, "Foundations of Ulysses: The theory of security," Odyssey Research Associates, Inc., Tech. Rep. RADC-TR-87-222, July 1988.
[20] J. Alves-Foss, "Mechanical verification of secure distributed system specifications," Ph.D. dissertation, Department of Computer Science, University of California, Davis, 1991.
[21] J. Alves-Foss and K. Levitt, "Verification of secure distributed systems in higher order logic: A modular approach using generic components," in *Proc. IEEE Symposium on Research in Security and Privacy*, 1991, pp. 122–135.
[22] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceeding of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sept. 1975.
[23] P. O'Connell, "The Idaho Partitioning Machine: A MILS partitioning kernel model in ACL2," Master's thesis, Dept. Computer Science, University of Idaho, Dec. 2003.
[24] *The Partitioning Kernel Protection Profile*, The Open Group, June 2003, draft under review.
[25] D. Greve, M. Wilding, and M. Vanfleet, "A separation kernel frmal security policy," in *ACL2 Workshop 2003*, 2003.
[26] J. Alves-Foss and C. Taylor, "An analysis of the GWV security policy," in *ACL2 Workshop 2004*, 2004.