The MILS Architecture for a Secure Global Information Grid

Dr. W. Scott Harrison, Dr. Nadine Hanebutte, Dr. Paul W. Oman, and Dr. Jim Alves-Foss
Center for Secure and Dependable Systems

Multiple Independent Levels of Security and Safety (MILS) is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The goal of the MILS architecture is to ensure that all system security policies are non-bypassable, evaluatable, always invoked, and tamper-proof. Using these formally proven security policies guarantees information flow control, data isolation, predictable process control, damage limitation, and resource availability. As applications are not considered trustworthy components, information flow control needs to be performed by entities external to the applications. This approach allows for the integration of legacy applications that do not necessarily have security integrated into them. Therefore, the MILS architecture creates an environment that adds safeguards to previously insecure applications, allowing the integration of possibly insecure applications into a secure environment. To accomplish this in the MILS architecture, guards are placed between communicating entities to act as message content filters and enforce information flow control. This article discusses issues concerning design and implementation of MILS components for message routing and guarding on a secure Global Information Grid facilitating net-centric warfare and defense.

High-assurance systems are used in environments where failure can cause security breaches or even a loss of life [1]. Examples include avionics, weapon controls, intelligence gathering, and life-support systems. Before such a system can be deployed, there must exist convincing evidence that it can support critical safety as well as security properties.

The avionics community has addressed the need for safety-critical systems by developing the DO-178B and DO-255 standards, which provide a set of guidelines for the design, analysis, and evaluation of system safety [2, 3]. Though adequate for the safety evaluation of airborne systems, neither is sufficient to address the security concerns of critical security systems such as those that protect national security. Such high-assurance systems require the rigorous specification and implementation requirements outlined in the Common Criteria (CC) [4].

The CC is a jointly developed evaluation standard for software that was created by a consortium representing the United States, United Kingdom, Germany, France, Canada, and the Netherlands. The purpose of the CC is to standardize evaluation of security features in software, which allows, for example, the comparison of different security solutions. The CC achieves this by providing guidelines for the design, analysis, and evaluation of critical systems defined at seven Evaluation Assurance Levels (EALs). The higher the assurance level, the stricter the requirements mandated by the CC. At the highest levels (EAL 5-7), the CC requires the use of formal methods, mathematical models, and proofs [1].

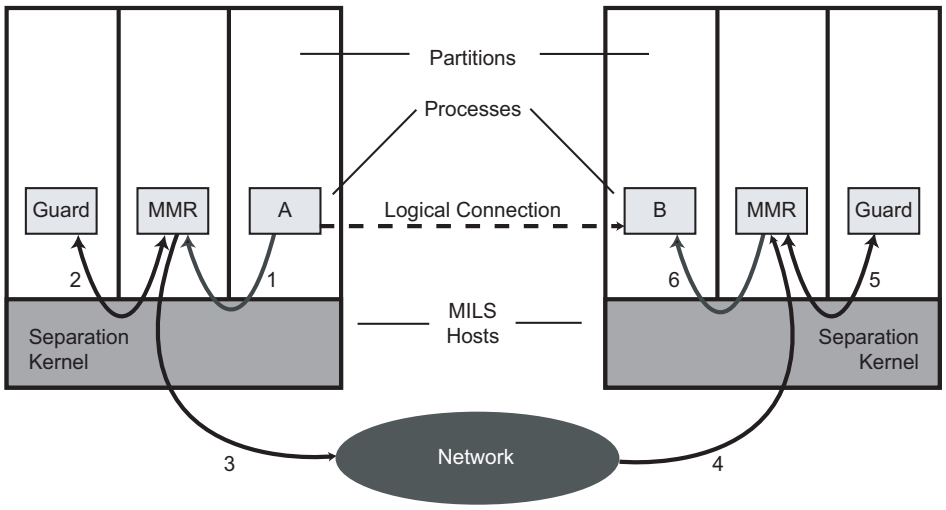
The level of difficulty and complexity of formal verification increases in an exponential manner with the number of analyzed lines of code (LOC). Code bases of over 100,000 LOC are considered to be unverifiable [5]. The goal for a verifiable software component is under 4,000 LOC [6]. With this restriction on code destined for EAL 5 certification or higher, the design shift from monolithic code bases to smaller modular components must occur.

A system designed specifically for EAL 5-7 certification is the Multiple Independent Levels of Security and Safety (MILS) architecture [7]. The MILS approach toward meeting the formal evaluation requirements of the CC is to separate system functionality into smaller, individually verifiable components. The MILS architecture enables the enforcement of system-wide information control policies via mechanisms built into the kernel as well as middleware components that create the authorized communications paths between applications.

One example of MILS middleware security component is an application-level message filter called a guard, or mediator. Since MILS guards can be verified independently with respect to other components, they can be built once and used within any MILS system that needs application-level message filtering.

This article describes the MILS initiative led by the Air Force Research Laboratory (AFRL) with stakeholder input from the Air Force, Army, Navy, National Security Agency, Boeing, Lockheed Martin, Objective Interface Systems, Green Hills Software, Lynux Works, Wind River, General Dynamics, Raytheon, Rockwell Collins, MITRE, and the University of Idaho. MILS technology is targeted toward the C-130 Avionics Modernization Program, F/A-22, F-35, C-130, Commanche, global positioning system, the Joint Tactical Radio System

Figure 1: *The MILS Architecture*



and the LandWarrior Program [8]. A prototype proof-of-concept of the system described in this article has been implemented within an embedded system at the University of Idaho.

This technology is essential for the Global Information Grid (GIG). The GIG is envisioned as a globally connected set of computer systems and software [9]. Object-oriented communications protocols (such as those we describe in this article) are vital to such a system [10]. Further, as the information on the grid will consist of many security classification levels, it will be absolutely necessary to control the information that flows through the GIG. The MILS architecture allows exactly that.

The MILS Architecture

MILS is a verifiable, secure architecture for executing different security-level processes on the same high-assurance system. The MILS architecture accomplishes this by providing two types of separation. MILS enforces a separation policy that strictly controls communication between processes of different security levels. This prevents, for instance, a top-secret process from communicating with an unclassified process. Further, MILS separates traditional kernel-level security functionalities into external modular components that are small enough for rigorous evaluation using formal methods. Verifiable secure systems can then be built from multiple, independently developed and certified components.

The foundational component of MILS is the separation kernel (SK). The SK segregates processes and their resources into isolated execution spaces called *partitions*. Processes running in different partitions can neither communicate nor infer each other's presence unless explicitly permitted by the SK. The SK enforces compliance to information flow policies via the MILS message routing (MMR) component. The primary function of the MMR is to route communication between applications in different partitions if that communication is allowed by the policies of the system [11]. If not, the MMR will not permit communication between the partitions.

In conjunction with the MMR, which simply fulfills routing functionalities on messages between partitions, guards enforce detailed, protocol-specific policies. A guard exists for each application-level protocol supported in a MILS system. If a guard determines that the content of a message does not comply with information flow policy, the guard will notify the MMR that will then disallow the communication attempt or take action based on security policy. Steps one through six in Figure 1 show the path of a message within a MILS architecture from the sending process (A) to the receiving process (B).

The advantage of using the MMR and guards is that the system does not have to trust the applications to conform to security policies. The MMR (and later, the guard) will enforce these policies. Thus, it is possible to have a secure MILS system while running untrusted applications within the partitions. This is because the SK prevents any other possible partition communication.

Because of the separation of responsibilities between message routing and message content filtering for each protocol, the individual components (the MMR and multiple guards) can be independently verified. Verification is only possible because the guards and MMR have distinct and well-defined functionalities. Neither accidental nor malicious communication attempts that violate system policy will be successful.

"The MILS architecture enables the enforcement of system-wide information control policies via mechanisms built into the kernel as well as middleware components that create the authorized communications paths between applications."

Many communication protocols were not designed to provide artifacts that allow systematic security policy violation handling such as proper error messages. In general, most protocols were not written with security as an objective, and thus, there are typically no error messages that are security-specific. As an example, consider two clients: a client classified as Secret requesting top-secret information, and a client classified as top-secret requesting the same information. Further, assume that there is an error from both clients in the request message (perhaps the query was incorrectly formed). Generally in this situation, a single error message would be returned to both clients. However, in many systems, this would not be correct behavior; we may not wish for the secret client to know that it was contacting a valid server at all, as this gives the client information about the state of the system that might invalidate security policy. Thus, there is a need, not generally implemented in most protocols, for security-specific error messages that do not reveal information about the state of a system.

Therefore, the MMR also has to determine which error messages are relayed back to the sender due to security policy violations without disrupting the overall execution of the communication initiator. A challenge similar to that of the proper relaying of error messages is that of integrating legacy software into a MILS system. This is because one of the design goals of a MILS architecture is transparency, i.e., existing (legacy) applications can be seamlessly integrated into a MILS environment. The MMR and communication channel guards facilitate this transparency.

Multi-Level Access Control

Multi-level secure (MLS) systems enforce a high-level, inter-partition security policy that dictates whether partitions with different clearances can communicate. Traditionally, the military model of a secure operating system includes a MLS concept. 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 data at multiple classification levels. Classic security models such as the Bell-LaPadula model [12] have been created to specify the secure behavior of such MLS systems. The problem with pure 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 labeled data and that there is no possible violation of the security policy. Even with a Trusted Computing Base architecture or reference monitor in place, there is often too much to evaluate.

The MILS architecture was developed to resolve the difficulty in certifying MLS systems by separating out the security

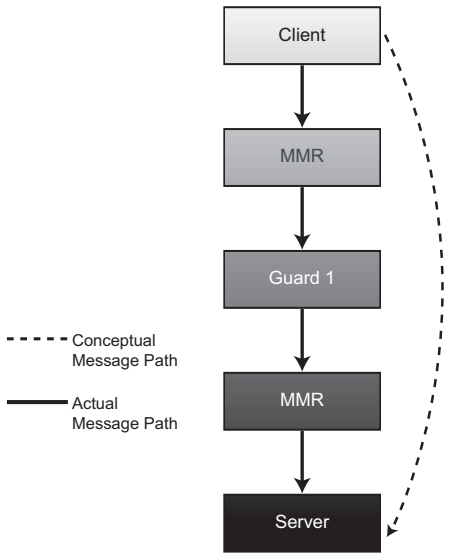
mechanisms and concerns into manageable components. A MILS system isolates processes into partitions, which define a collection of data objects, code, and system resources. These individual partitions can then be evaluated separately. This divide-and-conquer approach will exponentially reduce the proof effort for secure systems. For instance, a traditional MLS policy may allow a secret partition to send messages to a partition that has both *secret* and *top-secret* clearance. An additional application-specific transport policy (i.e., a protocol-specific guard) is needed to do this. The protocol-specific guard policy specifies constraints on the contents of messages sent between partitions already allowed to communicate by the MLS policy.

The MMR and SK can fulfill MLS policies, while the protocol guards enforce application-specific security policies that may or may not be MLS policies; however, the SK and MMR do work in tandem with the guard policies to provide fine-grained access control of application-level messages.

A MILS Database Server

Consider a multi-level secure database application. This database would contain entries of different security levels, e.g., top secret, secret, classified, and unclassified. Remote processes from different partitions can invoke read and write methods on this central database. However, if a client process that is only classified to handle *secret* data (e.g., Secret_Read()) attempts to invoke a read method on *top-secret* data (e.g., TopSecret_Read()) from the database server, the request must either be denied or the data must be downgraded from top-secret to secret prior to invoking the requested read.

Figure 2: *Processes, the MMR, and Guard Message Path*



Polyinstantiation is another solution to this problem; however, the challenges are similar. Stated simply, polyinstantiation is a situation in which users at different security classifications receive (possibly) different responses to the same queries. As an example (adapted from [13]), consider that we might have information regarding a ship (S), an objective (O), and a destination (D). When an unclassified user queries this information, he or she would receive the information {S=U.S.S. Starfish, O=Surveying, D=Hawaii}.

> *"Thus, it is possible to have a secure MILS system while running untrusted applications within the partitions. This is because the SK [separation kernel] prevents any other possible partition communication."*

However, a top-secret level user would receive the information {S=U.S.S. Starfish, O=Spying, D=Coast of Vietnam}, which presumably is the actual state of the system. Such a solution comes at the cost of having to create very large databases and, as above, requires authentication of the true originator of a request.

Adding functionality to the database partition to determine the true origin of the request sender and to verify that the sender has proper classification is a less complex solution. Such a solution, however, would cause other problems:

a. The server's code base might become too large to evaluate formally.
b. Dedicated server processes will have to be rewritten to account for every type of data transaction (e.g., top secret, secret, unclassified, etc.).
c. The server's responses to valid but unauthorized requests would need to be added to the server's code base.

The MILS solution is to allow the MMR to parse the message before it reaches the client, consult a policy, and then either pass or reject the message after an in-depth content analysis (which would potentially still be necessary, although not as thorough, for a system that incorporates polyinstantiation). Such an analysis would have to account for routing policies and protocol or content-specific policies. This requires extra complexity in the MMR, which increases with every application-layer protocol supported in the MILS system. Therefore, the MMR simply determines if the communication between partitions is allowed according to the security policy and, if so, identifies the message type and passes it on to the protocol-specific guard.

When a protocol guard receives a message from the MMR, it parses the message, consults a protocol-specific policy, and then notifies the MMR whether to allow or deny the message. If the message is allowed by the protocol policy, the MMR sends the message on to its destination. Otherwise, the MMR will perform error handling. Figure 2 shows the logical structure of the MMR and the protocol-specific guard situated between client and server applications.

To illustrate the previous example within a system that contains a MMR and protocol guard, assume that a secret level client attempts to invoke the read method of the top-secret database server. The client encapsulates the request in a protocol-specific message and sends it. The MMR determines that the client is allowed to communicate with the server, recognizes the message as being of a particular protocol type, and routes it to the appropriate guard for analysis. The guard examines the request message and determines that the client is not allowed to invoke the read method of the top-secret database object. Finally, the guard instructs the MMR to discard the message, and possibly generate an error message to be returned to the client.

It should be noted that simply discarding the message will generally be insufficient. As the client never receives a response from the server, the client will likely continue to make the same request. Since this condition also occurs under normal circumstances such as those due to temporary network outages, for example. Thus, one function of the protocol-specific guard is to generate error packets, which look like standard protocol error messages that will be sent back to the client. If no response is expected, obviously an *error* message need not be returned.

An Example Policy

The Common Object Request Broker

Architecture (CORBA) is a platform-independent middleware architecture that facilitates common client/server requests. CORBA Object Request Brokers communicate via the General Inter-Orb Protocol (GIOP). We will illustrate the MILS architecture with an example using a CORBA GIOP guard.

Figure 3 illustrates an example of the MMR and a protocol-specific guard (in this case, a CORBA GIOP guard) working together to enforce MLS and transport policies. Client A is a CORBA client application running in *secret*-level partition 1, and client B is a CORBA client running in *unclassified*-level partition 2. A CORBA database object is running in multi-level secure partition 3, which has both *secret* and *top-secret* clearances. The database object has two methods, Secret_Read() and TopSecret_Read(), which require the invoking client to have secret and top-secret clearances, respectively.

The MLS policy for our example system is that all processes can only communicate with processes of equal or higher security clearances. The GIOP transport policy extends the MMR's MLS policy by placing further constraints on the GIOP messages sent between partitions the MMR already allows to communicate.

As Figure 3 shows, the MMR allows client A to communicate with the database object because partitions 1 and 3 both have secret clearance. The GIOP guard, however, restricts client A's communication with the database object to only invocations of the Secret_Read() method since A does not have the top-secret clearance required to invoke TopSecret_Read(). Client B is not allowed to access the database object at all. The MMR blocks all requests sent by B because partition 2 does not have the equivalent clearance(s).

Conclusion

The MILS architecture provides a cost-effective and efficient way to build verifiable secure systems. By separating security functionality into modular components, high-assurance systems can be engineered and evaluated much more rapidly and independently. The MILS architecture is an approach to system design that is supported by industry and government. SKs, the lowest layer of the MILS architecture, are already being deployed by multiple real-time operating system vendors [14]. Common criteria protection profiles are currently being developed for both the separation kernel [15] and MILS middleware [16].

In this article, we have shown how a security policy can be enforced on GIOP messages sent between MILS partitions. A policy that allows or disallows method invocations based upon the requesting client partition, the servant object, and the object method also can be used. Our testbed implementations also show how guards can allow a MILS system to enforce both MLS and application-specific policies, thus providing fine-grained access control of inter-partition communication.◆

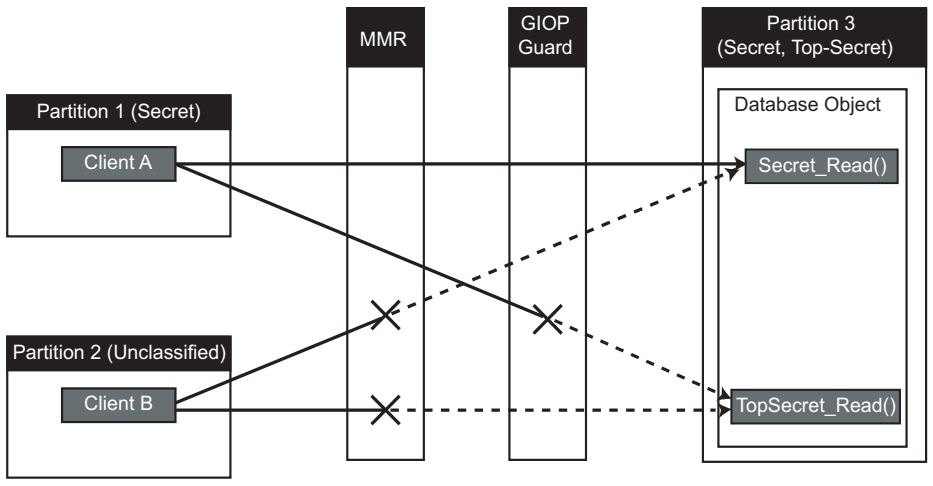
Clarification

This material is based on research sponsored by the AFRL and Defense Advanced Research Projects Agency (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.

References

1. Alves-Foss, Jim, W. Scott Harrison, Paul Oman, and Carol Taylor. "The MILS Architecture for High Assurance Embedded Systems." International Journal of Embedded Systems 2005 (to appear).
2. Radio Technical Commission for Aeronautics. "Software Considerations in Airborne Systems and Equipment Certification (RTCA DO-178B)." Washington: RTCA Inc., 1992 <www.rtca.org>.
3. Radio Technical Commission for Aeronautics. "Requirements Specification for Avionics Computer Resource (ACR) (RTCA DO-255)." Washington: RTCS Inc., 2000 <www.rtca.org>.
4. Common Criteria. CC Recognition Arrangement: Common Criteria for Information Technology Security Evaluation (Vers. 2.1). 2004 <www.commoncriteriaportal.org>.
5. Dransfield, Michael, et al. "MILS/MLS Architecture for Deeply Embedded Systems." NetCentric Operations 2004.
6. MacLaren, Lee. "New Options in Embedded Computing Security." Boeing Technical Excellence Conference, 2003.
7. Alves-Foss, Jim, Carol Taylor, and Paul Oman. A Multi-Layered Approach to Security in High Assurance Systems. Proc. of the Hawaii International Conference on System Sciences, 2004.
8. Adams, Charlotte. "Keeping Secrets in Integrated Avionics." Aviation Today Mar. 2004 <www.ghs.com/download/articles/Aviation_today.pdf>.
9. Miller, Alyson, Mark Jefferson, and Jeff Rogers. "Global Information Grid Architecture." The Edge: MITRE Advanced Technology Newsletter July 2001 <www.mitre.org/news/the_edge/july_01/miller.html>.
10. Ackermann, Robert. "Jointness Defines Priorities for the Defense Department's Global Grid." Signal Magazine Apr. 2001 <www.afcea.org/signal/articles/anmviewer.asp?a=120&z=115>.
11. Hanebutte, Nadine, et al. Software Mediators for Transparent Channel Control in Unbounded Environments. Proc. of the 6th IEEE Information Assurance Workshop, 2005.
12. Bell, D. Elliot, and Leonard LaPadula. "Secure Computer System: Unified Exposition and Multix Interpretation." ESD-TR-75-306. Bedford, MA: MITRE Corp., 1976.
13. Jajodia, Sushil, and Ravi Sahndu.

Figure 3: *An Example Policy*



Polyinstantiation Integrity in Multilevel Relations. Proc of the IEEE Symposium on Research into Security and Privacy, 1990 <http://citeseer.ist.psu.edu/121576.html>.

14 Ames, Ben. "Real-Time Software Goes Modular." Military and Aerospace Electronics. 14.9 (2003):24 <www.ghs.com/download/articles/GHS_RTOS_modular_090103.pdf>.

15. National Security Agency. U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness (Vers. 0.621). Washington: NSA, 2004.
16. University of Idaho. MILS CORBA Protection Profile, Vers. 0.52 (draft). Moscow, ID: Univ. of Idaho, 2003.

About the Authors

W. Scott Harrison, Ph.D., is an assistant professor in the Computer Science Department at the University of Idaho, where he has been since 1999. His current research involves information assurance and computer security issues. Harrison has a doctorate in computer science from Tulane University in New Orleans.

Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 84844-1008
Phone: (208) 885-4114
Fax: (208) 885-7099
E-mail: harrison@cs.uidaho.edu

Paul W. Oman, Ph.D., is a professor of computer science at the University of Idaho. He held the distinction of University of Idaho Hewlett-Packard Engineering Chair for a period of seven years. Oman is a senior member in the Institute of Electrical and Electronics Engineers (IEEE) and is active in both the IEEE and the IEEE Computer Society. He has published more than 100 papers and technical reports on computer security, computer science education, and software engineering. He is a past assistant editor of *IEEE Computer* and *IEEE Software*.

Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 84844-1008
Phone: (208) 885-4114
Fax: (208) 885-7099
E-mail: oman@cs.uidaho.edu

Nadine Hanebutte, Ph.D., is a postdoctoral fellow at the University of Idaho's Center for Dependable and Secure Systems. Her industrial experience includes work at the HypoVereinsbank in Munich, Germany as a software engineer for the risk-control department. Her current research includes software security, information assurance, and computer security education. Hanebutte has a Master of Science in computer science from Otto von Guericke University in Magdeburg, Germany, and a doctorate from the University of Idaho.

Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 84844-1008
Phone: (208) 885-4114
Fax: (208) 885-7099
E-mail: hane@cs.uidaho.edu

Jim Alves-Foss, Ph.D., is an associate professor of computer science at the University of Idaho. He also serves as the director of the University of Idaho Center for Secure and Dependable Systems, which focuses on information assurance education and research. His current research involves the use of formal methods for protocol analysis and security policy verification.

Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 84844-1008
Phone: (208) 885-4114
Fax: (208) 885-7099
E-mail: jimaf@cs.uidaho.edu