Middleware for security and safety critical systems

Gordon M. Uchenick describes an architecture designed to make high levels of certification achievable.

Whether you are designing a nuclear fusion ignition system, an advanced weapons system, an unmanned robotics vehicle or an integrated avionics suite, safety or security critical systems typically require certifications that their software content meets all of its requirements. Certifications require a lot of time, cost and schedule, especially in the most important applications where the consequences of failure are unacceptably high. Certification can also be an arduous task, often taking more time and costing more money than the original software development itself.

Embedded systems developers benefit from software architectures that make high levels of certification more achievable. One such architecture, Multiple Independent Levels of Security (MILS), is intended to simplify the certification process at high levels of assurance, making it practical and affordable. It also enables code and artifact reuse to leverage investment. MILS is a layering architecture with the primary goal of minimizing the amount of code that needs the most rigorous inspection. Its layers are the Separation Kernel, Middleware, and Application.

MILS approach to middleware

Traditional operating system architectures have a large body of code running in privileged mode. Because nothing constrains the behavior of privileged code, it can violate the system safety and security policies such as data isolation and controlled information flow. Considerable effort is required to rigorously prove all of this code is well behaved, that it does not violate or weaken policy enforcement. The time, cost, and risk of failure for this proof increases exponentially as the code base gets bigger and more complex.

The MILS architecture simplifies high assurance certifications by moving system functions from the Separation Kernel layer to the Middleware layer. The result is a Separation Kernel that is significantly smaller and simpler, much more conducive to certification at a high level of assurance. The increased confidence that the Separation Kernel will always enforce the system safety and security policies correctly reduces assurance requirements for the Middleware and Application layers.

The MILS approach is to move as much system code as possible to the Middleware layer where it runs in unprivileged mode. I/O services, file system and communications protocol stacks are large and complex code bases. Instead of being able to violate the system policies, all of that code is now strictly subject to them. Because of this fundamental difference, a basic to moderate level of rigor of inspection for that code is now appropriate. Basic to moderate level code evaluations are achievable, practical, and affordable for large code bases and have been done many times. MILS Middleware, typically linked with the applications that it supports, can be grouped into three general categories: System Services, the Partitioning Communications System, and Network Middleware.

System services as middleware

Embedded application developers are accustomed to using a full-featured real-time operating system (RTOS). Most of the code in an RTOS implements the complex services described above. The MILS Separation Kernel has a sparse application programming interface (API); it does very few things but it does them very well. An effective way to bridge this gap is to port an entire RTOS, a familiar environment friendly to the application developer, to run as Middleware in a user mode MILS partition. Although it may be perceived as a significant engineering task, the path to this transition is surprisingly straightforward.

One of the key objectives for any commercial RTOS is portability to a wide range of processors. For this reason, RTOS code is written to the greatest extent possible in a processor independent manner. Processors and the need for implementation mandates the RTOS designer to minimize the number of operations requiring privileged instructions because privileged instructions are processor specific. The bulk of RTOS code is processor specific. The need to perform processor specific activities is achieved by isolating the processor specific code into a few small modules. This set of modules is called the Hardware Abstraction Layer (HAL). RTOS developers are adept at writing processor specific activities and need to perform new HALs; it is how they can quickly and economically offer their platform on the latest and greatest processors. They are providing their standard RTOS with a new HAL.

If a HAL cannot abstract a physical processor, it can also abstract a MILS Separation Kernel. Instead of executing processor specific privileged instructions, a HAL can use system services to request these operations be performed on its behalf by the Separation Kernel. The high assurance Separation Kernel can be trusted to enforce the system safety and security policies as it executes requested system services and returns the results to the HAL. The HAL translates those results as necessary and then gives control back to the
RTOS. The net effect of this architecture is that a MILS node can contain its own “Guest.” Each partition in a MILS environment with minimal change. Software engineers enhancing legacy applications or developing new ones have the familiar RTOS environment and development tool set. Actual target hardware is often not available during the early to middle stages of a project, creating potential development and testing bottlenecks. If some of the embedded applications do not require special hardware, or if that hardware can be emulated, unit testing can be done on any COTS board supported by the RTOS.

**Network stacks as middleware**

Middleware often requires private execution and threading contexts. Network protocol stacks often fall into this category and are built to run in their own private MILS partitions. Strong separation is a significant benefit of this architecture. The application and the protocol stack are protected from each other, considerably reducing total system vulnerability. Fortunately, moving protocol stacks out of the kernel into their own partitions can be completely transparent to the applications that use them.

Let’s use TCP/IP as an example. The semantics of the socket library (or its equivalent for non-IP based networking) are unchanged from the application’s perspective. All modifications are hidden “under the hood.” A traditional socket library implementation of data transmission via send() copies data from the application to network buffers. The copy procedure is necessary because the scope (i.e., lifetime) of the data to be sent is unknown. Once the data is copied, it is handed over to the stack’s transport layer via either a function call or a system service, depending on how the stack was implemented.

Alternatively, a MILS implementation of the send() function uses the Separation Kernel’s tightly controlled information flow facilities to move outbound data from the application partition to the protocol stack running in its private partition – see figure 1. The Separation Kernel guarantees that there is no other way for the data to get there. The most important point is that the application just called send() in either case. What send actually does with the data is irrelevant to the application as long as the data gets to wherever it should be going.

**Partitioning communications system middleware**

All of the COTS components that are used to connect one machine to another are a potential source of vulnerability. The protocol stacks, interface cards, routers, switches and media were all originally designed to move data optimally. Safety or security may not have been original requirements and were often bolted on as a series of emergency patches issued as amateur hackers and professional attackers succeeded in penetrating distributed systems. This reactive approach is always one step behind, counteracting threats only after the damage has been done.

Why not start over, developing protocol stacks with safety and security as a key requirement? TCP/IP protocol stacks are large and complex bodies of code. This assertion is proven by noting that it took Richard Stevens three substantial volumes to explain what they do and how they do it in the masterwork *TCP/IP Illustrated*. Putting robust safety and/or security policy enforcement into a protocol stack is suboptimal because the resultant code body size and complexity conflict with the rigorous inspection necessary to verify enforcement is adequate.

The amount of code that must be trusted to enforce the security policy among distributed nodes can be significantly reduced in the same manner as MILS significantly reduces the amount of trusted code that enforces the security policy within a single node. MILS simplifies large problems by dividing them into small problems that can interact only in predictable ways – divide and conquer. This methodology can also be applied to distributed systems. A security policy enforcement module, the Partitioning Communications System (PCS), can be interposed between the application and protocol stack partitions instead of adding security functions to the stacks themselves. The PCS can perform its security policy enforcement functions transparently with respect to both the applications and the protocol stacks – see figure 2.

The policy enforcement functions performed by the PCS are:

- **Flow Separation:** Providing trustworthy separation of the individual flows (e.g., TCP port connections) on a shared medium (e.g., Ethernet). These flows are to or from multiple application partitions in the node local to the PCS. Because the separation is trustworthy, these flows can be at different security levels (i.e., TOP SECRET vs. SECRET) or different safety levels (i.e., CRITICAL vs. NON-CRITICAL). In the past, separate physical links were required to provide trustworthy “air gap” separation.

- **Strong Authentication:** Identifying the node, the application, and the application instance at the “other end” of each flow. High value data should not be transmitted until the sender can verify that the recipient is identified, authorized, and in a safety or security preserving state.

- **Bandwidth Provisioning and Partitioning:** Controlling the portion of total channel capacity that can be used by each flow. This useful security function enforces the designer’s targets for allocation of shared bandwidth which shortens system integration time and increases total system reliability. Denial of Service attacks from malicious or errant software in
the transmitting node are also suppressed.

- Covert Channel Suppression: Link level encryption is not enough. Significant characteristics of the data and the applications can be discerned by analyzing traffic patterns, especially in response to external events which may have been caused by the malicious observer himself. The PCS controls message length and timing to counter this threat, transparent to the applications.

- Secure configuration: One cause of failure a distributed system is errors or attacks that effectively get each node to enforce different versions of the overall safety or security policy. Because the different versions are not necessarily compatible and are also enforced by each other, each node can be correctly enforcing its local policy but total enforcement for the enclave can be compromised. The PCS validates that proper versions of the system policy are being enforced for each flow.

- Trusted Image Loading: It is a common requirement to either load or update applications “over the air” after a distributed system has been deployed. The PCS controls this operation, validating that the software originates from an authenticated source, that the software has not been modified, that the download or update itself has been authorized, and that the resources required by the downloaded module conform to the safety or security policy.

PCS communications policy enforcement has a very important characteristic which is required for high assurance. It is NEAT, an acronym for:

- Non-bypassable: The MILS Separation Kernel’s data isolation and controlled information flow guarantee that all data must pass through the PCS. It can’t be circumvented. The PCS guarantees that data will always flow through downgraders, guards, and firewalls, never around them.

- Evaluatable: The PCS is small enough and simple enough to enable the mathematical proof of correctness that is required for high assurance verification, certification, and accreditation. The “trusted plumbing” that the PCS provides simplifies distributed safety and security modules, enabling their high assurance evaluations and validations.

- Always Invoked: The safety and security policy is enforced each and every time that data is transferred. The distributed system does not transmit data unless and until all protection is in place.

- Tamper Proof: Trustworthy data isolation prevents erroneous or malicious applications from modifying the PCS code or the data that defines the safety and security policy. The PCS assumes that both the applications and the network are hostile, protecting itself from invalid structure or sequences in the data that it handles. Attacks based on modifying policy data can’t succeed.

How is this put together? Revisit figure 2. Application 1 and Application 2 need to communicate with their counterparts in an external system. It is important that their data flows not interfere with each other in either content or Quality of Service. Application 1 could be handling SECRET classified data and Application 2’s traffic could be unclassified; Application 1 could be safety critical and Application 2 could be of lesser importance to the total system. The PCS is interposed between these applications and the communications partitions in both the sending and receiving nodes. In this arrangement, the PCS, because of its high assurance, can be trusted to maintain the required separation and bandwidth allocations when the communications facilities can’t be trusted to do this by themselves.

Network Middleware

Network Middleware such as CORBA, DDS, and Web Services offer many features mostly aimed at providing location transparency. Applications aren’t required to know where their input data comes from and where their output data is going. This transparency conflicts with safety and security requirements. In safe and secure communications systems we need to know exactly where data comes from and exactly where it is going.

Why is porting Network Middleware to MILS desirable? MILS/PCS communications resolves the conflict. Legacy applications can continue to use their existing Middleware foundations and new applications can be developed in those familiar environments because MILS/PCS communications allows only authorized information flows to occur between familiar system configurations and existing applications.