Building Secure Systems using Model-Based Engineering and Architectural Models

Jörgen Hansson
hansson@sei.cmu.edu

Peter H. Feiler
phf@sei.cmu.edu

John Morley
jmorley@sei.cmu.edu

A system designer faces several challenges when specifying security for distributed computing environments or migrating systems to a new execution platform. Business stakeholders impose constraints due to cost, time-to-market requirements, productivity impact, customer satisfaction concerns, and the like. And users exercise power at the desktop over computing resources and data availability. So, a system designer needs to understand requirements regarding protected resources (e.g., data), confidentiality, and integrity. And, a designer needs to predict the effect that security measures will have on other runtime quality attributes such as resource consumption, availability, and real-time performance.

After all, the resource costs associated with security can easily overload a system. Security processing can increase usage of processing power, bandwidth, battery (in embedded systems), and other resources. Despite that, security is often studied only in isolation and late in the process. However, using model-based engineering (MBE) tools, the Carnegie Mellon® software Engineering Institute (SEI) has developed analytical techniques to model and validate security according to flow-based approaches and to represent standard security protocols such as Bell-LaPadula [Bell and LaPadula 1973, 1976], Chinese wall [Brewer 1989; Lin 1989], role-based access control [Ferraiolo 1992], Biba model [Biba 1977] for enforcing confidentiality and integrity.

The Bell-LaPadula model has been successfully used as a foundation to specify and analyze multi-level security (MLS). MLS captures the notion that a system processes data items classified at multiple security levels. The information flow security policy must thus prevent high-level classified data items not to be compromised into low-level objects (i.e., unintentional lowering of required security clearance.)

Security analysis using MBE tools allows software validation by identifying data elements to be protected, components that should be allowed access to those elements, and appropriate communications channels. This analysis permits the designer to enforce security at the minimum level required, use sanitization, and map software architecture to hardware that supports the required security levels—to take advantage, for instance, of the multiple independent levels of security (MILS) paradigm.

Key points:

- *Security is often studied in isolation and late in the development process.*
- *Security comes with a cost in terms of resource usage and bandwidth.*
- *MBE and the building of architectural models facilitate quantitative analysis of security and other dependability attributes.*
- *System designers can employ MBE techniques on architectural models to predict runtime behavior early in the development life cycle.*
- *Validation using MBE tools can be conducted at multiple layers and different levels of fidelity.*

PREVENT SYSTEM INTEGRATION PROBLEMS AND SIMPLIFY LIFE-CYCLE SUPPORT—AN MBE APPROACH TO SECURITY

Modeling of system quality attributes is often done—when it is done—with low fidelity software models and disjointed architectural specifications by various engineers using their own specialized notations. These models are typically not maintained or documented throughout the life cycle, making it difficult to predict the impact of change on attributes that cut across system functionality. The unanticipated effects of design approaches or changes are discovered only late in the life cycle, when they are much more expensive to resolve. Analysis of a system architecture model offers a better way to predict the behavior of the integrated components when assembled into the system.

The Architecture Analysis & Design Language (AADL) is a MBE tool that allows analysis

- using a single architecture model to evaluate multiple quality attributes
- early and often during system design or when upgrading existing system architecture
- at different architecture refinement levels as information becomes available
- along diverse architectural aspects such as behavior and throughput

Integration is a major cost and risk in complex systems [NIST 2002]. In a study conducted by NIST, it was observed that 70% of all defects are introduced prior to implementation, i.e., requirements phase and system and software design phase. Yet, only 3.5% of the defects were detected in these phases; 50.5% of the faults were detected in the integration phase. The defect removal cost ranged from 5x to 30x relative to the cost of removing the defect in the phase of introduction (if it had been detected). Other sources are reporting similar estimates, and while the numbers vary, the conclusions do not.

System understanding is a major cost driver during system maintenance. Model-based engineering is a proactive measure to determine the viability of a system architecture to enforce security. By providing a security validation framework using MBE and AADL, organizations benefit from multiple uses out of single architectural model annotated with additional properties capturing non-functional requirements and parameters used for validation, preventing many system integration problems and simplify the life-cycle support.

IMPACT AND TRADEOFF ANALYSIS

Key point:

- *Architectural decisions taken to assure confidentiality and data integrity cut across software, hardware, and their connections.*

Security is an architectural concern that intrinsically crosscuts through all levels of the system (application, middleware, operating systems, and hardware), requiring intra- and inter-level validation of security. Security also has immediate effects on the runtime behavior of the system, specifically other dependability attributes.

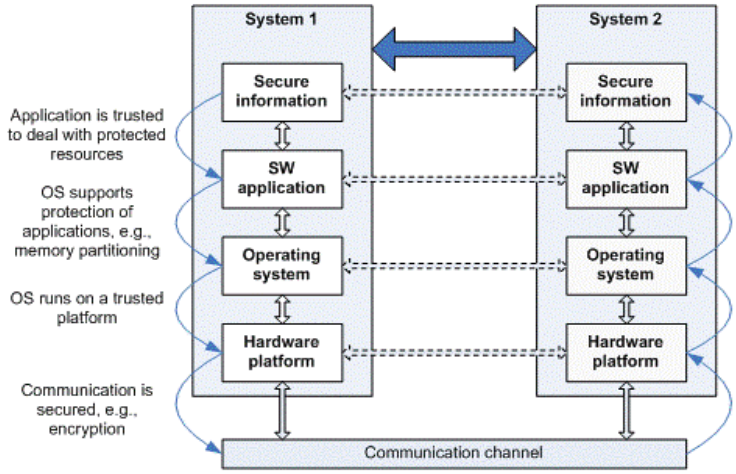

Fig. 1: System perspective on security

First, we need to enforce inter-level and intra-level security when designing an architecture and building the system. Fig. 1 depicts various system levels. For the purpose of this example, we focus on validation of security privileges against confidentiality requirements and thus assume authentication and other necessary security services are enforced. Ultimately, we want to ensure that the software applications do not compromise the confidentiality of the secure information they are exchanging. Consequentially, software applications need to execute on top of a secure operating system, mapped to a protected and secured hardware memory space, and communicate over a secure communication channel. Thus, if the data is confidential, then every layer needs have a clearance of at least that level. While the example is coarse in scale, it demonstrates that security needs to be enforced at every architectural level to ensure secure communication between the applications. Said differently, we must both inter-level and intra-level security.

Second, security comes with a cost. Encryption, authentication, security and protection mechanisms add increased bandwidth both in terms of CPU, network, and memory).

Indeed, these increases further affect temporal behavior of the system (worst-case execution time, response time, schedulability, and end-to-end latency) as well as power consumption (especially important in battery-driven, or limited-life time devices (e.g., sensor networks or portable communication devices). The system designer is facing the challenge of trading these quality attributes against each other and security can thus not be considered in isolation. This is a particular concern for embedded and real-time systems, which are characterized to operate under significant resource constraints while ensuring high levels of dependability (as reliability, availability, or safety) as well as security. Said differently, security is interlinked with the other non-functional behaviors such as predictability/timeliness, resource consumption, and inadvertently affect reliability and availability. (Some of these dependencies are depicted in Fig. 2 below).

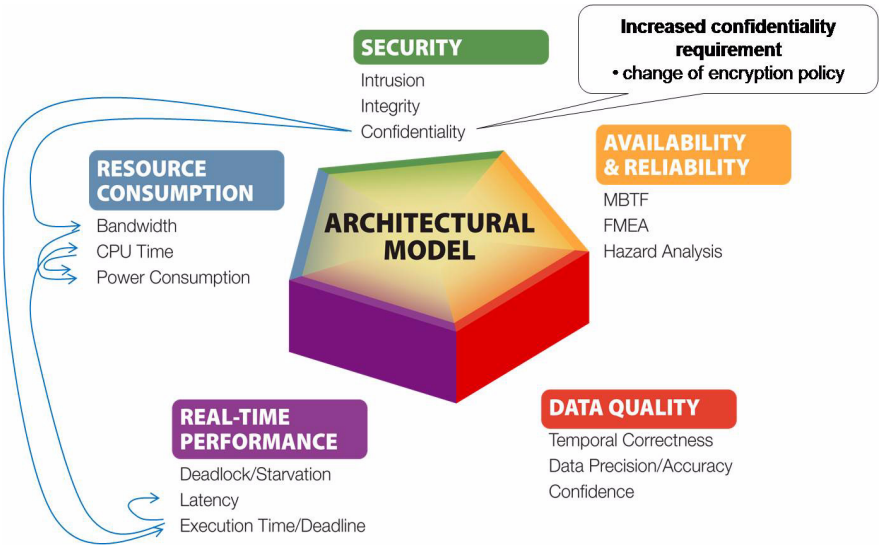

Fig. 2: A change in security level will have impact on other non-functional attributes

It is easy to see the value of a single architectural model by considering the scenario with a number of independently created models that is supposed to reflect the same system architecture (consistency across models and system)

It follows that any change to modeled architecture during its life time requires each of these models to be updated and validated that it correctly reflects the actual system architecture. Hence, it is difficult to consistently reflect any architectural changes in one analysis dimension in other dimensions, due to interaction among the dependability attributes. It becomes important, from the perspectives of economics, system correctness, system quality, and development life cycle) to integrate the different analysis dimensions into a single architecture model. A single-source architecture model that is annotated with

analysis-specific information of multiple types can drive model-based engineering by generating the analysis-specific models from this annotated model. This allows changes to the architecture to be reflected in the various analysis models with little effort by regenerating them from the architecture model. This approach also allows us to evaluate the impact across multiple analysis dimensions, allowing the designer to conduct adequate tradeoff analysis, evaluate different architectural variations prior to system realization, thus gaining confidence in the architectural design.

Models also serve a purpose in post-development phases, where models can be used to evaluate effects of reconfiguration and system revisions. A model-based approach allows validation at different levels (e.g., validation of confidentiality and integrity, validation of architectural patterns that enforce security, and consistency validation of architectural assumptions).

.

The SEI uses the **Architecture Analysis & Design Language (AADL)** to model and document system architecture and to provide a platform for multiple analyses. The AADL, an international industry standard, supports multiple analyses from a single architectural model, enables modeling and analysis throughout the life cycle, and provides analysis of runtime behavior (what) rather than functional behavior (how).

Through its XML/XMI interchange format, the AADL supports model interchange and tool chaining. The SEI offers the freely available **Open Source AADL Tool Environment (OSATE)** set of analysis plug-ins for performance, resource consumption, security, and reliability. AADL also can be used with (i) UML state and process charts, through its UML profile, (ii) the SEI Architecture Tradeoff Analysis Method® (ATAM®), to drill into root causes and develop quantitative analysis, and (iii) assurance cases, to support claims made about the safety, security, or reliability of a system.

For more information on MBE and AADL, see www.sei.cmu.edu/pcs and www.aadl.info.

Key Point:

- *An MBE approach lets the system designer see security considerations holistically—through the virtual integration of software and hardware.*

AN MBE APPROACH TO VALIDATING CONFIDENTIALITY

Confidentiality addresses concerns that sensitive data should only be disclosed to or accessed by authorized users (i.e., enforcing prevention of unauthorized disclosure of information). Data integrity is closely related as it concerns prevention of unauthorized modifications of data. The security framework, utilizing AADL and OSATE features:

- representation of confidentiality requirements of resources (i.e., objects);
- representation and generation of security clearance/privileges of subjects operating on the objects;
- representation of an access matrix, specifying allowed access operations of subjects on objects to support integrity

To model and validate the confidentiality of a system, we distinguish between general and application-dependent validation. General validation of confidentiality is the process of ensuring that a modeled system conforms to a set of common recommendations or design guidelines, expressed as a set of conditions that support system confidentiality independent of a specific reasoning framework for security. Those conditions should hold in the general case; as a result, they are necessary but not sufficient (i.e., satisfying the conditions indicates the system is viable for enforcing confidentiality). General validation of confidentiality assumes that subjects and objects are assigned a security level, that is the minimum representation to enforce what are commonly referred to as the basic confidentiality and need-to-know principles.

Key points:

- *The OSATE security plug-in analyzes and validates architectural models for confidentiality and controlled sanitization. among other aspects. New analysis techniques can also be added by adopters.*
- *MBE tools can be used to set and ensure that the least amount of privilege is enforced throughout the system.*

Application-specific validation refers to validating the system given detailed confidentiality requirements and a specific reasoning-based security framework. The following types of security validation and analysis, using OSATE, are currently available.

- **Basic confidentiality principle**—access should only be granted if given the appropriate security clearance.
- **Need-to-know principle**—access should only be granted access to a resource if there is a need. For example, a person having top-security document clearance should not necessarily be allowed access to all documents but only to those related to his or her function in a project. In our notation, this is captured in the composite security label consisting of class and category.
- **Controlled sanitization**—stipulates that lowering the security level of an object or subject should only be authorized and performed by a privileged subject.
- **Non-alteration of object's security requirements**—a subject using an object as input should not alter the security level of the object, even if the object is updated as an output from the subject. The rationale for this condition is that a subject can have a security clearance that exceeds the maximum required security level of an object. Increasing the security level of the object beyond its range implies that security requirements do not align between the subjects that operate on a dependent object. Thus, a subject with less security clearance than an object cannot continue its operation as expected.
- **Hierarchical condition**—ensures that (i) a component has a security level that is the maximum of the security levels of its subcomponents and (ii) all connections are checked to determine whether the source component of a connection declaration has a security level that is the same or lower than that of the destination component.

The principle of least privilege has been identified as important for meeting integrity objectives; it requires that a user (subject) be given no more privilege than necessary to perform a job. This principle includes identifying what the subject's job requires and restricting the subject's ability by granting the minimum set of privileges required. With the object's security requirements specified in an AADL model, the least amount of privileges for the subjects

can be generated in a straightforward manner by analyzing the security levels of all objects accessed by the subject. Given that the subjects' privileges are specified, a mismatch between the least privilege and what has been specified results in two possible cases:

1. The assigned privilege is insufficient—that is, it does not dominate the required least privilege (the subject has been given incorrect privileges, the object has been wrongly associated with a subject, or there is an unauthorized access).
2. The assigned privilege exceeds the minimum privilege, which either is unnecessary or a consequence of that the subject might be associated with other objects that have not yet been described in the model.

The versatile concept of subjects and objects, a notion introduced by Bell and LaPadula, where subjects operate on objects by permissible access operations (read, execute, append, write) enables us to model and validate security at both the software and hardware levels. At the software level we can view processes, threads, and software components as subjects and data objects are objects.

Determining the viability of a system given confidentiality requirements of data objects and security clearance by users, one can see the validation must include (i) validation of the software architecture followed by (ii) validation of the system architecture where the software architecture is mapped to hardware components. By mapping the entities of a software architecture (e.g., processes, threads, and partitions) to a hardware architecture (consisting of, for example, CPUs, communication channels, and memory) and the like enables us further to ensure that the hardware architecture supports required security levels (see Figs. 3 and 4).

Consider the scenario of two communicating processes, both requiring a high level of security as because the data objects requires secret clearance. Furthermore, the system platform in this scenario consists of a set of CPUs with hardware support for various algorithms that encrypt messages before network transmission. By modeling the system, we can represent and validate that processes and threads (now considered to be objects) can be executed (access mode) on CPUs (subjects) with adequate encryption support. Furthermore, we can validate that CPUs (objects) communicate data (access modes of writing and reading) over appropriately secured communication channels. In a similar fashion, we can enforce design philosophies saying that only processes of the same security level are allowed to co-exist within the same CPU or partition or that they can write to a secured memory.

Key points:

- *The MBE approach allows designers to assure that software applications execute on top of a secure operating system, map to a protected and secured hardware memory space, and communicate over secure communication channels.*
- *Analysis of security measures can be done early and throughout the development life cycle using architectural models and MBE tools.*

```
-- Property intended to be customized by modelers.
-- Parameterizes the security property definitions.
property set Security_Types is
  -- Military levels by default
  Classifications:
    type enumeration (unclassified, confidential, secret,
                      top_secret);

  -- This must be the first element of Classifications
  Default_Classification:
    constant Security_Types::Classifications =>
      unclassified;

  -- Default set of categories
  Categories:
    type enumeration (A, B, C, D);
end Security_Types;
```

Fig. 3: Specification of security levels

```
property set Security_Attributes is
  Class: inherit Security_Types::Classifications =>
    value(Security_Types::Default_Classification)
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  Category: inherit list of Security_Types::Categories =>
    ()
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);
  -- ...
end Security_Attributes;
```

Fig. 4: Architectural components to which security levels and requirements can be connected

VALIDATING MILS ARCHITECTURES

MILS architecture represents one approach to architecting secure systems, and we are going to use it as an example of architectural validation. The MILS approach is to modularize an architecture by practicing "divide and conquer," and it uses two mechanisms for this purpose:

- First, the MILS architecture utilizes partitions to isolate processes, and each partition defines a collection of data objects, code, and system resources. Each partition can thus be evaluated separately (AADL supports the modeling of partitions and virtual processors).

- Second, each partition is divided into the following layers (each layer is responsible for its own security domain and nothing else):
    - •Separation Kernel (SK), which is responsible for enforcing data isolation, control of information flow, periods processing, and damage limitation
    - •Middleware Service layer.
    - •Application layer

A MILS architecture supports MLS, and it is argued that the reduced complexity of the system, by the use of partitioning and separation, is conducive to improved efficiency of certification. The approach is to separate security mechanisms and concerns into the following type of components, classified based on the way they process data:

- SLS—Single-Level Secure component; only processes data at one security level
- MSLS—Multiple Single-Level Secure component; processes data at multiple levels, but maintains separations between classes of data
- MLS—Multi-Level Secure component; processes data at multiple levels simultaneously

Well-designed MILS system separates functions into separate partitions (location awareness). Thus, the MILS architecture builds on partitioning as one key concept to enforce damage limitation. Virtual machines have been recognized as a key concept for providing robustness through fault containment, and this mechanism provides time and space partitioning to isolate application components and subsystems from affecting each other due to sharing of resources. This architecture pattern can be found in the ARINC 653 standard [ARINC 653].

Key point:

- *An MBE approach is conducive to the validation concerns most significant to MILS.*

The validation concerns most critical to MILS and where the MBE approach is most conducive are:

- Validating structural rigidity of architecture, which includes enforcement of legal architectural refinement patterns of a security component into SLS, MSL, and MSLS. This decomposition can be applied to components, connectors, and ports. A component can be divided into parts using the product, cascade, or feedback decomposition patterns [McLean 1994; Zakinthinos 1996]. Confidence in validation of an architecture increases with the fidelity of the modeling; MBE analysis can be applied at different architectural refinement levels. In a MILS context, this corresponds to the level of decomposition (i.e., confidence) is achieved if (i) refinement patterns ensure that security is enforced, recognizing intra- and inter-level security requirements, and (ii) the architecture enforces the refinement patterns and thus provide structural rigidity.
- Architectural modeling and validation of assumptions underlying MILS, which include assumptions with respect to damage limitation and partitioning in MILS, and validation of separation in time (and space). AADL supports the modeling of partitions with virtual processors.

- Validating requirements specific to the NEAT characteristics and the communication system. MILS requires that its SK and the trusted components of Middleware Services are implemented so that the security capabilities enforce what is commonly referred to as the NEAT characteristics:
  - **N**on-bypassable—security functions cannot be circumvented.
  - **E**valuatable—the size and complexity of the security functions allow them to be verified and evaluated.
  - **A**lways invoked—security functions are invoked each and every time without exceptions. The reference monitor concept can be used by the system architecture to enforce this for critical applications.
  - **T**amperproof—subversive code cannot alter the function of the security functions by exhausting resources, overrunning buffers, or other forms of making the security software fail.

These requirements include strong identity among nodes within an enclave (a group of MILS nodes), separation of levels/communities of interest (need cryptographic separation), secure configuration of all nodes in an enclave, secure loading (signed partition images, secure clock synchronization), suppression of covert channels (bandwidth provisioning and partitioning) and network resources (bandwidth, hardware resources, buffers).

MILS (Multiple Independent Levels of Security/Safety) has been proposed as an approach to building secure systems [Rushby 1981]. A joint research program among academia, industry, and government, MILS is led by the United States Air Force Research Laboratory. Participants include the Air Force, Army, Navy, National Security Agency, Boeing, Lockheed Marin, Objective Interface Systems, Green Hills Software, Lynux Works, Wind River General Dynamics, Raytheon, Rockwell Collins, MITRE, University of Idaho, and other academic institutes.

The MILS approach is based on the notion that separation—thus, limiting the scope and reducing the complexity of the security mechanisms—contributes to more easily analyzable systems. MILS focuses on reducing security functionality to the following security policies: information flow, data isolation, periods processing, and damage limitation. Controlling the information flow guarantees that system components can communicate only via designated and approved communication channels and paths (countering the threat of bypass). Data isolation requires that an application can access only memory explicitly allocated for it. Enforcing a policy for periods processing guarantees that the processor is not compromised, because it requires CPUs to be scrubbed before partition switching. Together with deterministic CPU allocation, this counters the threat of side effects of authorized system usage to be used as a covert channel. Damage limitation requires that all failures be contained and recovered locally (i.e., failures must not be propagated to cause a failure elsewhere in the system and thus the system should be free of chain reactions).

Case Study:

Security Validation of FPGAs

Rockwell-Collins used the AADL technology and security plug-in tool developed by the SEI to enable the high-assurance handling of data from multiple sensors having varying levels of security, such as airborne imagery field programmable gate array (FPGA).

Typically, you use a high-assurance processor to securely tag variable input. An FPGA is powerful and fast. The rationale for Rockwell-Collins to deploy FPGAs is that it is deemed easier to develop applications on an FPGA, and it also reduces the cost and time to market. The FPGA can further be reprogrammed at runtime (e.g., to fix bugs), which can lower maintenance-engineering costs.

Because FPGA behavior is more complex, architecture-level definition and analysis are needed. Rockwell-Collins developed architectural models of the FPGA using AADL and used the OSATE security analysis tool to validate security and demonstrate the high-assurance potential of FPGAs.

SUMMARY

Model-based engineering and the building of architectural models, facilitate quantitative analysis of security and other dependability attributes. Validation can be conducted at multiple layers and different levels of fidelity. Confidence in security validation increases with the level of decomposition, given that (i) refinement patterns ensure that security is enforced, recognizing intra- and inter-level security requirements, and (ii) the architecture enforces the refinement patterns and thus provides structural rigidity.

- Evaluation of an architecture configuration with respect to impact on other non-functional attributes—e.g., to quantify increases in power consumption, bandwidth usage (network, bus, processor), and performance (latency, schedulability).
- Validation of architectural requirements necessary to enforce architectural requirements. For example, the MILS approach requires the architecture enforces damage limitation (containment of faults through partitioning) and separation in time and space.
- Reduction of the effort necessary for recertification efforts in the event of architectural changes.

The overall objective of a secure system implies that security clearances are given conservatively (as opposed to generously). To this end, we can analyze models to derive the minimum security clearance on components in the model. Or to put it differently, we can use the notion of subjects and objects to determine the minimum security clearance for a subject based on the requirements of the objects being accessed by the specific subject.

By also pointing out differences between actual security clearances and the minimum security clearance required, a system designer can evaluate how effective and tight security is. Finally, by providing mechanisms to ensure that sanitization is conducted within allowed boundaries, the designer is allowed to analyze and trace these relatively more threatening security risks, as since sanitizing actions are permitted exemptions of security criteria and rules, and as such should be minimized in the system.

REFERENCES

[Alves-Foss 2006]
Alves-Foss, J., Harrison, W. S., Oman, P., and Taylor, C. "The MILS Architecture for High-Assurance Embedded Systems." I*nternational Journal of Embedded Systems 2,* 3/4 (2006): 239–347.

[ARINC 2008]
ARINC Incorporated. *Avionics Application Software Standard Interface: ARINC 653 Standard Document*. http://www.arinc.com

[Bell and LaPadula 1973]
Bell, D. E. and LaPadula, L. J. *Secure Computer Systems: Mathematical Foundations (MITRE Technical Report 2547, Volume 1)*. Bedford, MA: MITRE Corporation, 1973. http://www.albany.edu/acc/courses/ia/classics/belllapadula1.pd6]

[Bell and LaPadula 1976]
Bell, D. E. and La Padula, L. J. *Secure Computer Systems: Unified Exposition and MULTICs Interpretation (MITRE Technical report ESD-TR-75-306)*. Bedford, MA: MITRE Corporation, 1976. http://csrc.nist.gov/publications/history/bell76.pdf

[Biba 1977]
Biba, K. J. *Integrity Considerations for Secure Computer Systems (MTR-3153)*. Bedford, MA: MITRE Corporation, April 1977.

[Brewer 1989]
Brewer, David D. C. and Nash, Michael J. "The Chinese Wall Security Policy," 206-214. *IEEE Symposium on Security and Privacy.* Oakland, CA, May 1–3, 1989. http://www.gammassl.co.uk/topics/chinesewall.html

[Ferraiolo 1992]
Ferraiolo, David & Kuhn, Rick. "Role-Based Access Control," 554–563. *Proceedings of the 15th National Computer Security Conference*. Baltimore, MD, October 13–16, 1992. New York, NY: ACM Press, 1992. http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf

[Lin 1989]
Lin, T. Y. "Chinese Wall Security Policy—An Aggressive Model," 282–289. *Proceedings of the Fifth Aerospace Computer Security Application Conference*. Tucson, AZ, December 4–8, 1989. http://ieeexplore.ieee.org/iel5/7100/19131/00884701.pdf

[McLean 1994]
McLean, J. "Security Models," 1136–1145. *Encyclopedia of Software Engineering 2*. New York, NY: John Wiley & Sons, Inc., 1994.

[NIST 2002]
National Institute of Standards and Technology. *The Economic Impacts of Inadequate Infrastructure for Software Testing (NIST Planning report 02-3, May 2002)*. http://www.nist.gov/director/prog-ofc/report02-3.pdf

[Rushby 1981]
Rushby, J. "Design and verification of secure systems," 12-21. *Proceedings ACM Symposium on Operating System Principles, vol. 15.* Pacific Grove, CA (IUSA), December 14–16, 1981. http://www.csl.sri.com/users/rushby/papers/sosp81.pdf

[Zakinthinos 1996]
Zakinthinos, A. "On the Composition of Security Properties." Ph.D. dissertation, University of Toronto, March 1996.

[Zhou 2008]
Zhou, J. and Alves-Foss, J. "Security Policy Refinement and Enforcement for the Design of Multi-Level Secure Systems." *Journal of Computer Security 16,* 2 (2008): 107–131.