Boundary Flow Modeling

Dr. Richard Neely
Märzen Group LLC
35 Technology Way, Millyard Technology Park – Suite2W3
Nashua, New Hampshire 03060
Email: rbn@marzen.com

***Abstract* — Boundary flow modeling (BFM) is a method of modeling information security constraints and behavior of a system element — the system, its subsystems, and its components — in terms of information flows associated with each element.**

The development of BFM has been motivated over a number of years by the need to model security attributes in a distributed system environment. The BFM approach is in contrast with state-oriented modeling. The feature of "state" cannot reasonably be associated with a distributed system.

Within the BFM scheme, modeling is expressed in terms of relationships among information flows at interfaces appearing in the external boundaries of elements. The flow across an interface in a boundary is expressed as a *history* of the information entities that have flowed across the interface up to some point in time. These histories are the building blocks of BFM. As a whole, a BFM model of a system consists of relationships among sets of histories associated with each system element. Such relationships express the element's security attributes.

BFM also keeps track of dependency relationships among system elements nested (or layered) within a system. These relationships express logical dependencies among the security attributes of the elements. For example, asserted flow-modeled security constraints of lower-level components can be used to demonstrate logically that a higher-level component meets its own security constraints. This in particular contributes to addressing the security composition problem.

The BFM approach has been used for a number of distributed systems, including an internet gateway in its network context; a file system in the context of an operating system; a major weapon system; and a major modeling and simulation warfighter training system.

As the development of BFM progressed, we began to focus on three areas of improvement: more sophisticated methods of relating flow histories; integration of flow-based models with state-based models; and tool support for BFM. This paper concludes by summarizing those areas of BFM development.

***Keywords* — security modeling, security composition, data flow, distributed system, system state**

I. Introduction

Boundary Flow Modeling (BFM) is a method of abstracting and representing information security constraints and behavior of systems — especially distributed systems. BFM has been developed and applied by a group of security engineers (see Acknowledgements) over the past 25 years to support our efforts in providing security assurance for a variety of systems. The functional targets of this modeling scheme have included specialized systems, networks, and their subsystems and components.

BFM expresses these element characteristics in terms of information flows associated with each element. This paper explains the advantages of BFM for some kinds of systems; the mechanisms of BFM; and issues with the BFM approach whose solution is under way.

II. The Need for BFM

Traditionally, security modeling — including policy and design modeling — has been based on the state of the modeled target.

This is effective for many kinds of targets, for example operating systems and even entire platforms, such as workstations. But in other cases, state-based modeling has shortcomings. It does not work well for networks, or distributed systems in general, where "state" cannot be defined. Also, when a modeling approach is limited to state modeling, security integration — security composition — has been seen as difficult or impossible. A composition approach using BFM was presented in [ArchModeling]. Spafford ([EnsuringSecurity]) has pointed out that there has been a lack of research in this area. Neumann noted in [ComposableSystems] that abstraction layering would be an effective composition approach. Bell pointed out in 2005 ([LookingBack]) that to solve the composition problem, models must be interface oriented; but he also wrote that there were no such models.

The development of BFM was motivated by the need to effectively model security attributes in a distributed system environment. The BFM approach is in contrast with state-oriented modeling. In a distributed environment, flow-oriented modeling of constraints and behavior has several advantages. First, local views of the model are possible, regardless of the size and complexity of the system. Second — because of the existence of local views — security composition is not such a daunting problem. Third, depending on how BFM is customized for a particular system (or certain parts of it), the system's behavior can be modeled as nondeterministic, which is typically the case for a distributed system.

IV. How BFM Works

BFM has been progressively developed in the process of applying it to various systems over the years. In those systems, BFM has been used to model multiple element layers and logical relationships among them. Where needed, BFM has been shown capable of supporting higher assurance Common Criteria (CC) [CC] modeling requirements. BFM has been presented in a number of publications and has been used for several medium- and high-assurance systems, including:

- A multilevel secure (MLS) network gateway — Multinet Gateway — sponsored by Rome Air Development Center (RADC) and National Security Agency (NSA) (1985-90)
- A specialized file server for the Current Endorsed List Tools Example (CETLE) program sponsored by the National Computer Security Center (NCSC) (1991)
- Multiple platforms for a major weapon system, sponsored by the Air Force (1992-1999)
- A major modeling and simulation warfighter training system

A simple example. The application of BFM is now explained in terms of a very simple system, termed "Data Sorter," which is a high-to-low security guard (see Figure 1). The Data Sorter consists of three components: a Splitter and two data Senders, one for classified and the other for unclassified data. This example is used to show the method of flow-based modeling.

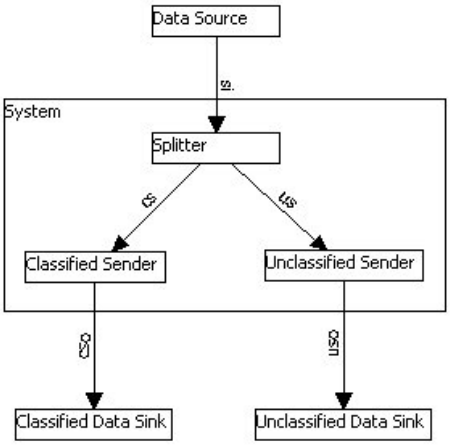

Figure 1. The Data Sorter, Its Components, and Its Environment

Security constraints are specified for each system element (i.e., the Data Sorter — "System" in the figure — and each of its components and peers). What is to be demonstrated about the Data Sorter is the claim that if the components and peers of the Data Sorter satisfy their security constraints, then the Data Sorter will satisfy its security constraints.

Developing a BFM-based model consists of several phases. These involve describing:

1. System structure
2. Data flows within the system
3. Security constraints — also termed assertions — of each element. These are expressed in terms of relationships among the data flows across element interfaces.
4. Logical relationships between security constraints associated with elements at different layers
5. An entire chain of logic, expressing the logical relationships between system-level security constraints and security constraints of leaf nodes in the system structure

We now elaborate these steps in developing and verifying a model. These are partially depicted in Figure 2.

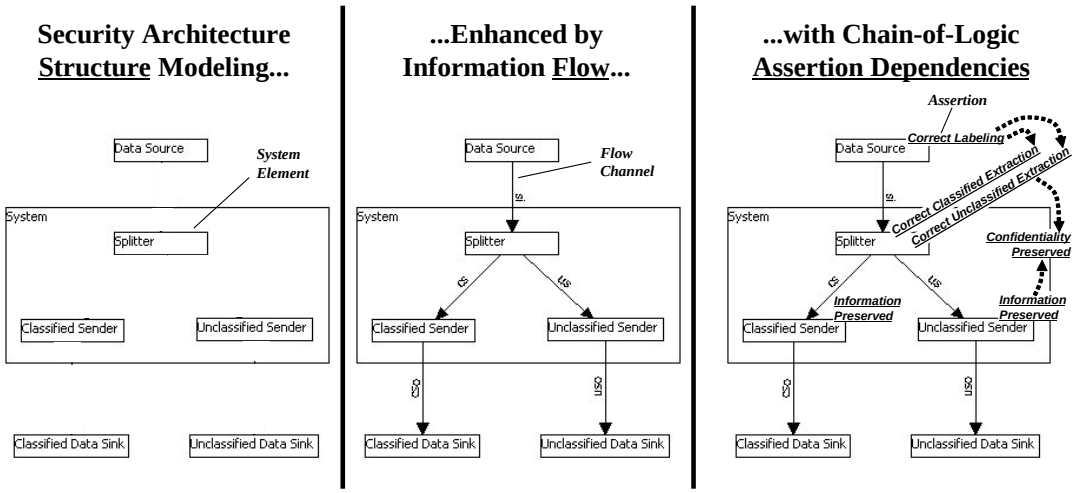

Figure 2. BFM Construction Phases

The *first* phase of the BFM process is to express the system structure (architecture). This consists of describing the system itself and related elements (subsystems, components, and environment). This is effectively a tree structure, which may be re-entrant in the case of re-used functionality. In this and the other phases, the content of the model precisely reflects development artifacts.

In the example (leftmost pane of Figure 2), the elements are System (Data Sorter), Splitter, Classified Sender, Unclassified Sender, Data Source, Classified Data Sink, and Unclassified Data Sink. Splitter, Classified Sender, and Unclassified Sender are subordinate to Data Sorter. The identification of the elements and their component relationships constitute the structure of the system.

The *second* phase of modeling involves extraction of data flow information from development documentation. Data flows as modeled are unidirectional (from an interface of one element to the interface of another) flow channels. Two-way flows, or broadcast flows, are broken down into unidirectional flows.

In the example (center pane of Figure 2), the data flows are named with the labels si, cs, us, cso, and uso.

The *third* phase consists of expressing the security constraints of each element in terms of related flow histories (i.e., data flow histories at interfaces of the element). Basic to the expression of a security constraint is a method of relating two or more flow histories. The ideal way of doing this is to define a relation — not a function — whose parameters are the flow histories. This avoids forcing determinism on the model of the constraint. This is important for a distributed system.

The "language" in which a security constraint is expressed is flexible: depending on project specifications and the nature of the system, a constraint may consist of a simple English sentence; a semi-formally expressed predicate calculus statement; or a formal mathematical expression.

In the example (rightmost pane of Figure 2), the modeled security constraints are as follows:

- Data Source constraint: *Correct Labeling*
- Splitter constraints: *Correct Classified Extraction*, *Correct Unclassified Extraction*
- Unclassified Sender constraints: *Information Preserved*

The *fourth* phase involves stating and verifying inferences: claims that the security constraints of a particular element are met, assuming that the security constraints of related elements (typically subordinate components and elements in its immediate environment) hold.

In the example (rightmost pane of Figure 2), the heavy dotted arrows depict inferences among the modeled security constraints.

The *fifth* and final phase of this method is documenting the "chain of logic" for the security constraints: applying the basic rule of inference (*modus ponens*) to the claims of the fourth phase, with the goal of showing that the security constraints of the system hold if the security constraints of the leaf elements hold.

V. BFM IMPROVEMENTS

In developing and employing BFM, we have begun focusing on three areas of improvement:

A. Providing logically sound relationships among architecturally separated flow histories in the distributed environment
B. Integrating BFM and state-based models within a distributed system
C. Establishing a production-quality tool to support BFM

These areas are discussed in the following paragraphs.

A. Providing Logically Sound Relationships among Architecturally Separated Flow Histories

Care must be taken to express the relationships among flow histories so that they are logically sound, particularly when the corresponding elements architecturally separated. For example, a relationship may be expressed that tacitly assumes that the contents of two histories being compared are being viewed at the same point in time. Yet a system-wide time referent is not possible within a distributed system.

In applying BFM, we have for example defined the function *Derived_From*, whose two parameters are two interface flow histories that are to be related. This function is used as part of the expression to model the security constraints of various elements. Informally, *Derived_From* (*h1*, *h2*) asserts that every entity within *h2* is "accounted for" by some entity within *h1*. We have typically defined the function as follows:

> For every entity *e2* in *h2*,
> there is an entity *e1* in *h1*
> such that *e1* = *e2*.

While the informal characterization of *Derived_From* is usable as a basis for representing a model, its more precise definition in terms of entities within histories is not categorically sound. For example, the selected entity *e1* may appear in *h1* at a later time than the entity *e2* appears in *h2*, and so *e1* cannot account for *e2*. Providing time stamps for entities within a history is not a solution because of the impossibility of a universal time referent. We have hypothesized defining *Derived_From* in a distributed context in terms of a similar function *Derived_From_Local*. The latter function would be applied only to "non-distributed" elements (e.g., platforms) and so for that function entities in histories could be compared time-wise. *Derived_From_Local* would be further defined so that it would "pass on" accountability information to *Derived_From* instances to be applied to higher-level, distributed elements.

In all cases, *Derived_From* makes no reference to the *order* of the entities in a history. In a distributed system with its characteristically nondeterministic behavior, it is not reasonable to assume that order will be preserved from inputs to outputs.

B. Integrating BFM and State-based Models

Certain system elements (operating systems, localized platforms, etc.) are best modeled using a state approach. Such an element may be a component within a distributed system.

Modeling of operating systems has matured considerably over the years, most recently with the Multiple Independent Levels of Security (MILS) approach ([MILS], [MLSwithMILS]). [SecureSharing] discusses a path to composability using the CC Protection Profile approach within "local" systems (e.g., platforms). BFM complements this, providing multi-layered composability above the platform level

To provide this complement — for security integration of an operating system within a distributed system — a means of providing security integration between state modeling and BFM is necessary.

We have taken a step toward this integration, at the separation kernel (SK) level. The approach was to make a careful comparison of a state model of an SK and a BFM model for that SK. More precisely, this involved:

- Understanding the claim made by the Greve, Wilding, Vanfleet (GWV) Formal Security Policy [GWVpolicy] for a separation kernel (SK)
- Expressing that same policy in BFM terms
- Demonstrating that the BFM claim is true if the GFW claim is true

In that way, when the SK is integrated into a larger system, the BFM expression of its policy can be used to support the validity of security constraints of higher level elements. We use "dual modeling" as a term to describe this approach.

This experiment in dual modeling at the SK level is only a prototype of the concept. Additional work must be done to further demonstrate the concept's feasibility. It is envisioned that dual modeling would typically occur at the platform interface (rather than an SK interface). That is because state modeling for an entire platform could reasonably be done, but could not be done for higher-level elements.

C. Establishing Tool Support for BFM

Having developed BFM features and applied BFM to various systems over the years, we have concluded that tool support for BFM is necessary. Tool support is advantageous in a number of ways. First, a support tool makes the modeling process more efficient than manual development of a model. Second, the presentation of the model to other developers, reviewers, and customers is more effectively accomplished using an appropriate tool. Third, accurate validation of security claims is more certain with a tool.

We have attempted the development of a tool from scratch. Even such a "rough and ready" tool was helpful in our efforts. Figures 1 and 2 in this paper were produced by the tool. Nevertheless, we concluded that a production-quality tool is needed. It is likely that extending a Unified Modeling Language (UML)-based development tool (such as Rational Rose) is the best direction to move in, and whenever we attempt BFM modeling of another system we plan to incorporate such a tool.

VIII. ACKNOWLEDGEMENTS

The development of BFM, its application to various systems, and any successes enjoyed by the modeling approach are the result of efforts by a number of colleagues, including Jim Freeman, George Dinolt, Michael Krenzin, and Max Heckard.

IX. REFERENCES

[ArchModeling]
Richard Neely, "System Security Integration Through Abstract Architecture Modeling," *InfoSeCon*, May 2006.

[CC]
Common Criteria for Information Technology Security Evaluation, version 3.1, http://www.commoncriteriaportal.org/cc/.

[ComposableSystems]
Peter G. Neumann, "Achieving Principled Assuredly Trustworthy Composable Systems and Networks," Computer Science Lab, SRI International.

[EnsuringSecurity]
Gene Spafford, "Exploring Common Criteria: Can it Ensure that the Federal Government Gets Needed Security in Software?", Testimony before the House Government Reform Committee, Subcommittee on Technology, Information Policy, Intergovernmental Relations and the Census, September 2003.

[GWVpolicy]
Jim Alves-Foss and Carol Taylor, "An Analysis of the GWV Security Policy," Fifth International Workshop on the ACL2 Theorem Prover and Its Applications, November 2004, http://www.cs.utexas.edu/users/moore/acl2/workshop-2004/.

[LookingBack]
David Bell, "Looking Back at the Bell-La Padula Model," *Proceedings of the 21st Annual Computer Security Applications "Conference (ACSAC)*, December 2005.

[MILS]
W. Mark Vanfleet, Jahn A. Luke, R. William Beckwith, Carol Taylor, Ph.D., Ben Calloni, Ph.D., Gordon Uchenick, "MILS: Architecture for High-Assurance Embedded Computing," *CrossTalk*, August 2005.

[MLSwithMILS]
Rance DeLong, "MLS with MILS?", Naval Postgraduate School Center for Information Systems Security Studies and Research lecture, Santa Clara University, March 2006.

[SecureSharing]
Carolyn Boettcher, Rance DeLong, John Rushby, Wilman Sifre, "The MILS Component Integration Approach to Secure Information Sharing," *27th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, October 2008.