Integration of Formal and Heuristic Reasoning as a Basis for Testing and Debugging Computer Security Policy

J. Bret Michael

Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439

Edgar H. Sibley
David C. Littman

George Mason University
4400 University Drive
Fairfax, Virginia 22030

Abstract

Errors can arise in defining and evaluating computer security policy as well as in translating computer security policy into procedures. The effect of such errors in policy upon the secure operation of information systems can impose unacceptable levels of risk from the perspective of procurers and users of information systems. Relying on computer security paradigms based solely on formal methods makes it difficult if not impossible to detect and/or reason about certain classes of threats to computer security and vulnerabilities of information systems to these threats, especially for those aspects of information systems that are more readily amenable to modeling via non-formal methods. We present a paradigm integrating formal and heuristic reasoning as a basis for testing for and debugging computer security policy. To illustrate our approach, and to support our arguments, we consider the problem of reasoning about the plans of an agent who may be trying to compromise the security of an information system.

1 Introduction

Advances in information technology have resulted in shifts in computer security paradigms as well as in computer security theory. A *computer security paradigm* is a convention or template for representing and reasoning about computer security, whereas a *computer security theory* is a plausible or scientifically accepted principle offered to explain computer security phenomena in an information system context. It is possible for a shift in paradigm to occur while the underlying theory remains unchanged, and vice versa.

Recent advances in distributed computing technology, for instance, have lead to both new computer security theories and paradigms. For example, Wilkes [10] envisions the need for new theories and paradigms to address distributed computing architectures founded upon the concept of secure enclaves.

> [T]he natural organization of a business firm would appear to offer scope for keeping sensitive information within the confines of a particular computer or computer system. Each system would, in fact, form a secure enclave connected to other systems by links along which information could be passed from inside the enclave ...[with] the links ...connected to servers dedicated to the purpose [in a client-server system].
>
> Information transfer from one secure enclave to another can be caused to happen either by algorithm ...or by a person operating within that enclave.

Existing theories and paradigms, specifically those based upon centralized computing concepts, are not necessarily adequate for planning for and ensuring secure distributed processing of information. For example, what does the simple security property* mean in terms of object-oriented client-server† technology? As demonstrated by Jajodia and Kogan [3], an object in the object-oriented sense can take on the role of a Bell-LaPadula (BLP) Model [1] subject *or* object.

Similarly, do existing computer security paradigms provide necessary and sufficient support for modeling and reasoning about client-server information sys-

*A subject s may have read access to an object o if and only if $C(o) \leq C(s)$, where C is the security class.

†A client-server architecture is one of many possible classes of architectures from which to implement a distributed information system.

tems, in which the architecture incorporates object-oriented constructs such as objects, messages, and single or multiple inheritance?

Sibley, Michael, and Sandhu [7] argue that it is incumbent upon users of a computer security paradigm or adherents to a computer security theory to understand the assumptions underlying the paradigm or theory, respectively. For instance, one of the assumptions made in the BLP Model is that information is stored, labeled, and retrieved as containerized files. A containerized file, however, is an inappropriate data structure for use in modeling and reasoning about objects stored, labeled, and accessed at multiple levels of granularity (e.g., document, section, paragraph, sentence, word, byte, and so on); that is, for some objects, there do not exist "natural" analogs to containerized files in, for instance, a multilevel secure (MLS) database management system (DBMS).

2 Formal Methods

One view of the process by which computer security policies are transformed into information systems is as follows: computer security policies are defined, evaluated, and then translated into procedures [8]. During this process, errors can be introduced into computer security policy. Policy is often stated in a natural language (e.g., English) and policy semantics are context dependent. Imprecision in policy definition contributes to the introduction of, for example, inconsistent, unintended, or unsound policies. Errors in the definition and evaluation of computer security policy become embedded in an information system if they are not detected and resolved prior to mapping policy to procedures.

Formal methods have been introduced into computer security paradigms as a means for understanding policy and managing the complexity involved in representing and and reasoning about secure information systems. Formal methods provide a basis for systematically and mathematically representing and reasoning about security policies and procedures, irrespective of whether the policies and procedures are to be performed manually or automated.

Formal methods cannot be used to model and/or reason about certain classes of errors introduced into information systems. For example, it is impossible to determine if computer security policy is complete for any information system. McLean [4] proved that it is possible to derive a non-secure information system that does not violate the axioms set forth in the BLP Model. The BLP Model is incomplete in that it ignors information system internals such as the raising of low-level system inputs via information processing.

Dobson et al. [2] contend that formal methods are *not* sufficient for representing and reasoning about technological and social aspects of computing. For example, the BLP Model cannot capture negotiations between BLP subjects resulting in granting, revoking, and delegating permissions, roles, responsibilities, obligations, and so on, specified in computer security policy. That is, interpretation and enactment of computer security policy is a *sociotechnological* issue. Dobson et al. suggest that some degree of reliance on "non-formal" methods, such as models based upon conversations between two parties (e.g., speech acts), is a prerequisite to understanding and managing both the sociological and technological aspects of intra- and inter-enterprise computing.

3 Should Not Happen Assertions

Although formal methods are not applicable for representing and reasoning about all facets of computer security policy, formal methods can assist us in precisely articulating and analyzing "should not happen" (SNH) assertions, one of many ways in which security policies can be formulated. For example, the following is a statement of the simple security property as a SNH assertion:

> *A subject s* **should not** *have read access to an object o if the security class of s does not dominate the security class of o.*

This SNH assertion corresponds to anticipated and unanticipated actions of information system users to access information classified above their clearance level. The simple security property, here stated as a policy, is intended to discourage users from performing actions resulting in unauthorized access to information; that is, a policy is intended to influence behavior, whether it be a human or a computer proxy for a human (e.g., a computer process). The procedures in an information system for implementing this policy are intended to both discourage, check for, and prevent unauthorized access to information.

Testing for and debugging errors in computer security policy requires some level of both formal *and* heuristic reasoning. For example, Michael [5] demonstrated that the ability of a resolution-style theorem prover to detect logical inconsistencies between composed security policies is dependent upon heuristic reasoning about how to complete linkages between and

disambiguate policy axioms; heuristic reasoning about domain information is applied in structuring policy axioms to guide the theorem prover in its search for logical contradictions between policy axioms.

Furthermore, it is not possible to model all of the possible inputs to and outputs generated by an information system explicitly. Consequently it is not possible to determine whether a set of SNH assertions is complete with respect to outcomes directly or indirectly resulting in the transition of an information system into or out of a secure state (i.e., a state in which computer security policy is not violated). Wahlstrom [9] describes the application of new technologies as a process of trial and error, arguing that it is difficult to predict the behavior and outcome of actions of automated systems and humans because technological systems interact with an unpredictable socioeconomic environment.

There are trade-offs to be weighed in deciding whether to apply formal or heuristic reasoning in testing and debugging computer security policy. Heuristic reasoning produces conclusions, whereas formal reasoning yields formal proofs. The risks associated with operating a secure information systems may dictate the construction of formal proofs that errors do not exist in a set of computer security policies and/or their counterpart procedures embedded in an information system. However, conclusions rather than proofs must suffice when formal methods cannot be applied.

4 Integration of Formal and Heuristic Reasoning

We propose a computer security paradigm based upon the integration of heuristic and formal reasoning. In this paradigm, heuristic reasoning is used to provide intermediate testing and debugging of policy, in support of formal methods. Consider the following scenario:

> *Suppose a person is observed simultaneously quacking like a duck and entering data into a MLS DBMS from a compartmented mode workstation; two minutes later the person abruptly stops quacking like a duck while continuing to type at the keyboard.*

In what ways can quacking like a duck contribute to security violations? Does this sequence of observed actions provide us with an indication as to whether the MLS DBMS transitioned into or out of a secure state? Are there explict SNH assertions in place addressing the observed actions and their outcomes? If not, were the actions and/or outcomes unanticipated?

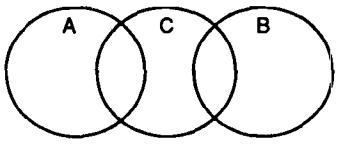
One of the problems that arises in answering questions about plans—sequences of actions devised and/or enacted by an actor to achieve a goal—and plan outcomes is that they are not always observable or inferable, such as the creation and use of a covert channel. For example, a user may perform actions after regular business hours or behind an office partition, making observation difficult or unlikely. Similarly, in some cases previously observed actions may not have been recorded for future reference. Without a record of past observations, it may be difficult to infer goals and/or likely outcomes of the actions, especially if pattern matching is to be used in analyzing the actions. Consequently, gaps in our knowledge of or ability to observe or infer plans can result in an incorrect policy, unsound policy, or incomplete policy, that is, errors in the coverage of anticipated and unanticipated plans and plan outcomes.

The actions of the user quacking like a duck may not have been anticipated by the person or persons observing the user's actions. The user's plan may be difficult to determine. For example, the user's actions may appear to the observer to have no distinct pattern from which to infer the goal(s) behind of the sequence of actions. If the plan or its outcome cannot be observed or derived, there is little if any basis upon which to determine whether SNH assertions cover the user's plan or the plan's outcome.

Figure 1 shows a categorization of SNH assertions. Assuming that it is not possible to observe or infer the plan or its outcome, we can only deduce that the plan and outcome fall in the areas delineated by A or B.[‡] In this diagram, we know that plans and outcomes contained in the areas $A \cap B$ or $B \cap C$ are covered by SNH assertions. The SNH assertions are incomplete if there exist any plans or plan outcomes in areas $A - C$ or $B - C$. SNH assertions contained in the area defined by $C - (A \cup B)$ are unsound in the sense that these assertions do not correspond to possible plans and outcomes. Incompleteness and unsoundness indicate errors have been introduced during policy definition, policy evaluation, or policy mapping. A summary of each area in the Venn diagram is summarized in Table 1.

The acceptable level of risk that an information system will transition into one or more non-secure states due to an unanticipated sequence of actions or unmodeled SNH assertions will vary among users or procurers

[‡]This is an example of the completeness problem; we make a tacit assumption that $A \cup B \cup C$ is the Herbrand Universe.



A: Anticipated plans and/or known outcomes
B: Unanticipated plans and/or unknown outcomes
C: Should not happen assertions
A U B: All possible plans and outcomes

Figure 1: Venn diagram

Table 1: Summary of error types by area

Area	Error Description
$A \cap C$	**No error**: there exist SNH assertions covering all plans and outcomes in this area
$B \cap C$	**No error**: there exist SNH assertions covering all plans and outcomes in this area
$A - C$	**Error**: all plans and outcomes in this area are *not* covered by SNH assertions
$B - C$	**Error**: all plans and outcomes in this area are *not* covered by SNH assertions
$C - (A \cup B)$	**Error**: there exist SNH assertions in this area that do not correspond to possible plans and outcomes

of information systems.

Heuristic reasoning, based upon the knowledge of previously observed behaviors and their outcomes and heuristic rules founded upon domain knowledge, can be used to reason about computer security policy. Rather than representing computer security policy explicitly, testing and debugging can be performed upon plans. Michael et al. [6] explored a modeling paradigm for representing intentions in information systems. Specifically, they attempted to model the state of an actor, with respect to computer security policy, as a triple: Is the actor *ready*, *willing*, and *able*?[§] For instance, for the time interval over which the person intermittently quacks like a duck, he or she is ready, willing, and able to violate computer security policy.

Suppose the following heuristic rules are at the disposal of the person responsible for testing and debugging computer security policy:

Heuristic Rule 1 *A sequence of actions that distracts users working at compartmented mode workstations can result in a sudden and temporary reduction in computation, input, and output operations performed by a MLS DBMS.*

Heuristic Rule 2 *Sudden and temporary reductions in computation, input, and output operations performed by a MLS DBMS can be used to create a covert channel.*

Relying on current observations of the users at other workstations and the two rules derived from past observations, one conclusion we can make is that the user's goal in quacking like a duck over different intervals of time is to pass classified information to unauthorized parties, that is, to create and use a covert channel. Based upon this conclusion, we could propose the definition and evaluation of the following new policy (SNH assertion):

Policy 1 *A person shall not perform actions that can be observed by and potentially disturb users, while at their workstations, of a MLS DBMS.*

This policy can now be axiomatized so that formal methods can be applied in testing and debugging the policy in the context of a secure information system.

In summary, some of the reasons why formally modeling the behavior of actors is difficult include the following:

[§]Computer processes and other inanimate objects are assumed to always be in a willing state; they have the volition of the person or persons who created them.

- Definition of goal states is imprecise.
- Precise definitions of what constitutes "computer security policy" or "computer security" is confounded by the circumscription problem.
- Security policies may be implicit (e.g., known to actors but not explicitly represented in manual or automated records) or depend upon unstated or unknown (to the modeler) domain knowledge.

Heuristic reasoning can be used to either predict outcomes for "what-if" scenarios or reason about an observed scenario; that is, heuristic reasoning supports both proactive and *ex post facto* threat and vulnerability analysis.

Moreover, the task of reasoning about computer security policy and its implications is ill-structured, such as in

- Understanding and managing the interface between policy and requirements.
- Evaluating the intent underlying actions.
- Applying formal methods with limited domain knowledge.
- Observing behavior from which to predict future behavior, only when conditions permit.
- Determining whether action sequences remain within a certain bound.

Our hypothesis is that effective use of heuristic reasoning can lead to the effective generation of SNH assertions; that is, heuristic reasoning can be used to identify and model domain knowledge in support of formal methods.

5 Testing for and Debugging Errors

We envision testing for and debugging computer security policy taking place at system design, compile, and run-time. At *design-* and *compile-time*, structural and static checking are performed, respectively. At run-time, however, plan checking is performed and is based upon policy dynamics; that is, changes in policy interaction and introduction of real-world knowledge. Upon completion of a particular check, computer security policies (and consequently the procedures implementing security policies) can be modified to effect desired changes in the behavior of a system or its users if unanticipated outcomes occur and are not covered by existing SNH assertions, and/or SNH assertions are violated.

The four phases of testing and debugging computer security policy are shown in Figure 2.

1. policy knowledge base building,
2. plan compilation,
3. plan execution and monitoring,
4. correction of the policy knowledge base on the basis of unexpected outcomes or SNH assertions; that is, feedback to phases (1) through (3).

Formal methods can be applied during phases (1) and (2), whereas heuristic reasoning is required during phases (3) and (4).

6 Computer Support for Modeling and Reasoning About Plans

Some aspects of testing and debugging computer security policy are readily automated. Sibley *et al.* [8] describe a *policy workbench* as a set of integrated computer-based tools for assisting users in defining policy, evaluating policy, and mapping policy to procedures.

Figure 3 depicts the flow of data among components of a hypothetical policy workbench architecture. We are currently exploring this and other candidate policy workbench architectures. In this architecture, the following information is stored in a knowledge base: (1) formalized policies and domain knowledge, (2) abduced and executed plans, and (3) formal deductions and heuristic conclusions. All of the components, represented as annotated boxes in the diagram, rely on the information in the knowledge base to carry out their functions. The knowledge base is updated to reflect newly performed observations and inferences; that is, testing and debugging of policy is an iterative and dynamic process. There is an underlying assumption that organizations modify their computer security policies over time in response to actual or predicted changes in the environment in which their information systems operate.

7 Summary

Modeling paradigms based solely on formal methods are not adequate for representing and reasoning

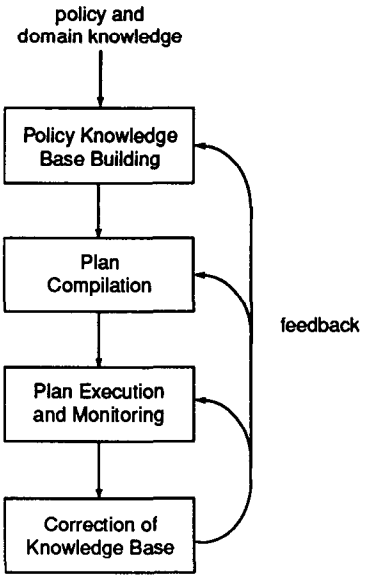


Figure 2: Four phases involved in testing and debugging computer security policy

about all aspects of an information system or computer security policy. Heuristic reasoning assists modelers in dealing with ill-structured aspects of computer security policy. We are exploring the coupling of formal and heuristic representation and reasoning techniques, with the goal of improving the state-of-the-art in defining and evaluating computer security policy, as well as translating computer security policy into procedures.

Acknowledgements

Richard Wexelblat actively participated in research meetings which produced some of the ideas which were subsequently further refined and presented in this paper. This work was not funded through Argonne National Laboratory.

References

[1] Bell, D. E., and LaPadula, L. J., Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, The MITRE Corporation, Bedford, Massachusetts, March, 1976.

[2] Dobson, J. E., Blyth, A. J. C., Chudge, J., and Strens, M. R., "The ORDIT Approach to Requirements Identification," *Proceedings of the Sixteenth Annual International Computer Software and Applications Conference.* Los Alamitos, California: IEEE Computer Society Press, 1992, pp. 356–361.

[3] Jajodia, S., and Kogan, B., "Integrating an Object-Oriented Data Model with Multilevel Security," *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy.* Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 76–85.

[4] McLean, J., "The Specification and Modeling of Computer Security," *IEEE Computer* 23, 11 (1990), pp. 9–16.

[5] Michael, J. B. A Formal Approach to Testing the Consistency of Composed Security Policies, Ph.D. dissertation, School of Information Technology and Engineering, George Mason University, 1993.

[6] Michael, J. B., Sibley, E. H., and Wexelblat, R. L., "A Modeling Paradigm for Representing Intentions in Information Systems," *Proceedings of the First Workshop on Information Technologies and Systems.* Massachusetts Institute of Technology Sloan School of Management, Cambridge, Massachusetts, 1991, pp. 21–34.

[7] Sibley, E. H., Michael, J. B., and Sandhu, R. S. "A Case-Study of Security Policy for Manual and Automated Systems," In *Proceedings of the Sixth Annual Conference on Computer Assurance.* IEEE

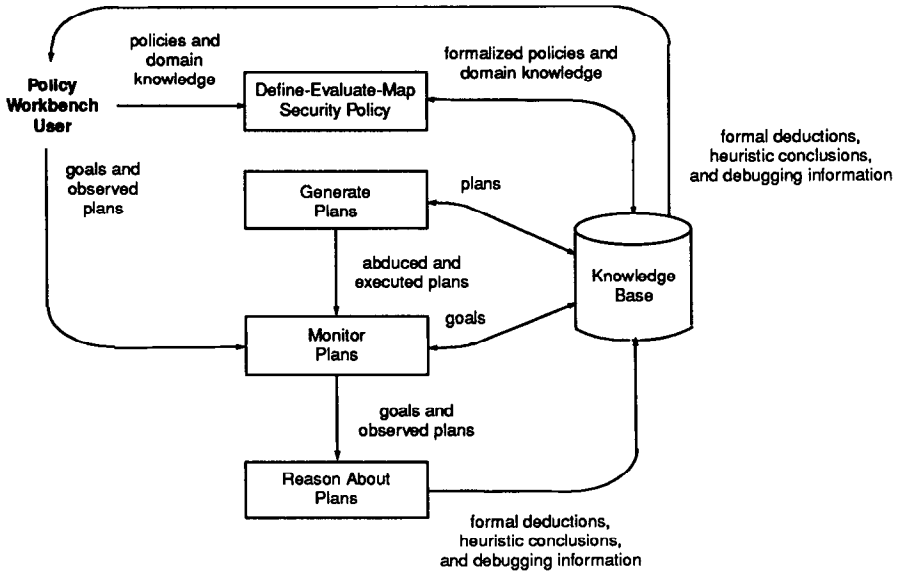


Figure 3: Data flow among components of a policy workbench

Computer Society Press, Los Alamitos, California, 1991, pp. 63–68.

[8] Sibley, E. H., Wexelblat, R. L., Michael, J. B., Tanner, M. C., and Littman, D. C., "The Role of Policy in Requirements Definition," *Proceedings of the IEEE International Symposium on Requirements Engineering.* Los Alamitos, California: IEEE Computer Society Press, 1993, pp. 277–280.

[9] Wahlstrom, B. "Avoiding Technological Risks: The Dilemma of Complexity," *Journal of Technological Forecasting and Social Change* 42, 4 (1992), pp. 351–365.

[10] Wilkes, M. V., "Revisiting Computer Security in the Business World," *Communications of the ACM* 34, 8 (1991), pp. 19–21.