New Paradigms for High Assurance Software

John McLean
Naval Research Laboratory
Code 5543
Washington, D.C. 20375

Abstract

We present a new paradigm for the development of trustworthy systems. It differs from our current paradigm by separating distinct desiderata that are bundled in the *Trusted Computer System Evaluation Criteria*, requiring that our formalisms be tied to real world concerns, requiring a uniform method for assuring that formalisms are met, replacing a code-then-validate methodology by a refinement-based methodology, and using composability logic to develop systems from COTS software.

1 Introduction

Anyone presented with our current paradigm for producing trustworthy systems, as, e.g., presented in [5], would wonder how the paradigm relates to the properties we would really like our systems to have. Nowhere in [5] is there a discussion of why desiderata are bundled the way they are, how properties and techniques for verifying that systems possess these properties are supposed to drive up the cost of penetrating a system, or how we can produce systems that satisfy the criteria in a cost-effective manner. This paper examines the current paradigm and presents a new paradigm for producing trustworthy systems that is derived from considerations of what we would like to have from our systems. We then put forth specific research proposals to implement the required paradigm shift in four areas: trust analysis, system property and specification development, refinement methodology, and composability logic.

2 Our Current Paradigm

Our current paradigm for producing secure systems, as exemplified by [5], consists of trying to specify some ideal of security, for example, access control,* and, depending on the level of trust required, spend varying amounts of money assuring that the ideal policy has been implemented in the system. At this point in the process, covert channel and penetration analyses are performed.

One obvious problem with the current approach is its exclusive focus on confidentiality: it contains no integrity or availability requirements. A second problem is that its security levels are too coarse-grained. As we move from lower to higher evaluation levels within it, functionality requirements (such as auditing), confidentiality requirements, and assurance requirements all increase. It is unclear why increases in functionality, confidentiality, and assurance should be bundled.† What is most unclear is why complete assurance is not required at every level. It is hard enough to find a sequence of properties $P_1, \ldots, P_n$ such that penetrating a system with P_{i+1} is costlier than penetrating a system with P_i, even without having to incorporate considerations reflecting the fact that the likelihood that a system actually has property P_i may differ from the likelihood that it has property Pi+1. More to the point, there is no reason to assume that a security problem missed after having spent $\$n$ on demonstrating that it has some property P_i will cost substantially less to exploit than one that is missed after having spent \$2n on demonstrating that it has P_i. Yet, the distinction between each level of [5] primarily represents a difference in the cost of producing a system. More specifically, it represents the money to be spent verifying the system's faithful implementation of an access control policy.

Another problem with the current approach is that it breaks down even for extremely high assurance systems. Access control models, such as Bell and La-

*In fact, if one were to strictly follow [5] there is virtually no leeway, either concerning the ideal (access control) or the method for specifying the ideal (a state machine model with certain properties).

†They are, in fact, not bundled in the ITSEC [4].

1993 ACM 0-89791-635-2

Padula (BLP) [2], do note preclude the possibility of covert channels in systems that conform to them even when explicitly supplemented to include restrictions on changing security levels [12]. Although Noninterference [6] does a better job with respect to storage channels, it fails to detect timing channels and to protect upgraded input [13]. Further, its application is restricted to deterministic systems, and, more seriously, to deterministic specifications. This makes it all but unusable for many real systems. Nondeterministic versions of Noninterference based on possibilistic trace models, such as Nondeducibility [17] and Restrictiveness [10], address only nonprobabilistic (i.e., noise-free) storage channels and still fail to protect upgraded input [13]. These models leave the detection of probabilistic storage channels and all timing channels to a later stage in system development. Although there are models that eliminate all channels from systems that conform to them, for example FM and PNI [7,13], and techniques for proving that systems satisfy these models [8], these models and verification techniques are still in the research stage.

The problem with any model that leaves the detection of (some class of) covert channels until after system coding is that the cost of eliminating any channels detected at this stage of software development can be prohibitively expensive. This stems from a variety of reasons: (1) the improvement in hardware and the increase in multiprocessor architectures that permit the construction of extremely fast (e. g., over 750,000 bits/second timing channels), (2) the fact that eliminating covert channels can require an entire architecture to be redrawn, and (3) the fact that making changes to any computer system is vastly (75 times or greater) more expensive after code has been produced than during the specification phase [3]. It is ironic that one of the early motivations for using formal methods was as a cost saving measure. Formal specification, by supporting early error detection, was supposed to drive down development costs. By leaving the detection of a large class of security flaws until the end of the development process, the cost advantage of using formal specifications is greatly reduced.

The problem is not simply that models such as BLP and Restrictiveness are not perfect, but that once we have proven that a system satisfies one of these models, we don't know what we really have. We can be confident that a system that satisfies BLP is secure with respect to access control, but we know nothing about covert channels—their presence, their capacity, their ease of exploitation, the type of data at risk, etc. Restrictiveness addresses only noiseless storage channels. Like BLP, it gives us no information at all about the channels that may remain—timing channels and probabilistic storage channels. Since a noisy channel can very easily have a higher capacity than a noiseless channel, we can conclude very little about our system. In fact, it wasn't until recently that techniques even existed for computing the capacity of noisy timing channels [16].

This problem is compounded by the fact that many refinement methodologies do not preserve the properties specified in our models [9]. Functionally correct implementations of possibilistic models, such as Nondeducibility or Restrictiveness, do not necessarily preserve the security properties of these models. Although there is, at least, one refinement technique that preserves confidentiality requirements, viz. one based on call-based trace specification [1,11], application of this method to security is still in the research stage [14,15].

3 A New Paradigm

In an ideal world where the cost of security technology is negligible, we would field only systems that could provably satisfy our most stringent security requirements. However, in any ideal world that is obtainable, we must take into account that assurance can come only at a cost. In such a world, we should be able to determine the value of information that is at stake in a computer system, the resources at a penetrator's disposal, the cost of implementing various types of security properties, the cost incurred by a penetrator breaking into systems with those properties, and the cost incurred by an agent learning the information in a way that does not necessitate breaking the system. For example, when buying a lock, one must take into account the value of the goods being protected by the lock, the type of intruders we are concerned about (e. g., professional thieves or curious children), the cost of the various locks available, the expense incurred by someone successfully breaking the various locks, and the cost of gaining entry without breaking the lock (e.g., by bribing the key keeper). Research whose task is to satisfy these desires falls under the province of trust analysis.

Not only must we be able to compute the relevant costs, we must be able to specify and build systems that fit our needs as determined by our cost calculations. That is, for any dollar figure, say n, we should like to be able to specify a system that would cost $n to break and less than $n to build. (How much less will, of course, depend on the likelihood that somebody will

attempt to break into the system.) We must also be prepared to accept the fact that we shall want some systems to be unbreakable.

When we turn from simple locks to computers, things become more complicated. We must first be able to specify a variety of trust types and a variety of security properties that enforce these trust types. Roughly, each trust type t would correspond to the resources a penetrator could be expected to expend trying to break a system that contained the information, and the corresponding set of properties Pt would be sufficient to guarantee that it would cost more to break the system than a penetrator would be willing to spend. However, there is no reason to assume that these types will be linearly ordered. We may be interested in a system whose confidentiality is very hard to break (although not unbreakable), whose confidentiality can be broken only by leaving a trail, whose integrity is unbreakable, and whose availability can be compromised for only short periods of time.[‡] There is no obvious dominance relation between such a system and one whose confidentiality is unbreakable but which can be unavailable for long periods of time. Obviously, we must also develop methods for showing, in some sense, that Pt is the correct set of properties for trust type t and verification techniques to show that a system designed to process information of type t satisfies the set of properties Pt.

Once we have an adequate set of security properties, we must be able to build a system that implements specific security requirements with high assurance at a reasonable cost. We believe that this can be accomplished only by developing formal methods that allow us to specify and reason about all security-relevant aspects of system behavior (including time) and that allow us to reason about compositions of specifications. The latter ability will allow us to use commercial off-the-shelf (COTS) software, making assurance cost-effective. It will require us to limit ourselves to user interface specifications, as opposed, e.g., to state machine models that discuss implementation constraints.

We cannot leave out the cost of developing software, however, since COTS software may not meet certain requirements and since even though the cost of COTS software will be distributed over many systems, it does not come for free. We can no longer build systems and then look for security flaws in the completed system. Experience shows that the changes necessary to correct security flaws found in completed systems are often too expensive to make. We must move from a develop-and-validate methodology to a refinement based methodology where specifications state all system properties that are required and programs are written in such a way that we know that they satisfy these properties.

[‡]A paradigm possibly to follow here is cryptography where we settle for encryption that is computationally expensive to break rather than encryption that is unbreakable and where authentication issues are separated from confidentiality issues.

4 How do We Get There

Comparing where we would like to be with our current practice with respect to high assurance software, we see several major needs:

1. We need methods for quantifying the value of the information stored on a system;
2. We need to develop sets of properties that will drive up in predictable ways the cost of breaking system security in various ways;
3. We need methods for predicting the cost of implementing these properties within a system;
4. We need methods for specifying and verifying perfect confidentiality (including probabilistic and timing channels);
5. We need methods for formalizing non-confidentiality security properties such as integrity and availability;
6. We need methods for specifying less-than-perfect security;
7. We need methods for evaluating the appropriateness of the security properties we formulate in (2);
8. We need validation methods that are not limited in the sorts of availability, integrity, probabilistic, and timing properties they can prove;
9. We need validation methods that can handle larger programs;
10. We need validation methods that enable us to validate software down to the machine code level;[§]
11. We need validation systems that are, themselves, high assurance systems;

[§]This is actually insufficient if we are concerned about high-assurance *systems*. We stop at the machine code level in this paper only because we are limiting ourselves to high-assurance *software*.

12. We need methods that address the composability problem with respect to specification and verification;

13. We need validation methods that catch security flaws before they are too expensive to correct;

14. We need a larger stock of specified, trusted components from which to build trusted systems.

Needs (1)–(3) represent limitations for system security in general, not just for security of high assurance systems. Some of the needs can be met by research alone; others require research accompanied by experimentation and experience. All the needs stem, in some sense, from the fact that much of the local research that has taken place up to now in computer security has lacked an accurate, global conception of the ultimate goal.

Given the above considerations, we suggest replacing the current methodology by a new paradigm of system development. To implement this paradigm, research needs to focus on four areas: trust analysis, specification, code development, and validation. These areas are described in turn.

With respect to the trust analysis, we must determine how much protection various types of information deserve, what sort of attacks we wish to protect information from (e.g., confidentiality violations that depend on compromising a reference monitor, confidentiality violations that depend on covert channels, denial of service attacks, integrity attacks, etc.), and what sort of system properties will provide this protection. The relation between information value and system properties can be found only through experience, but it is a necessary research direction that must be explored, yet has heretofore been ignored, if we are to have a security development methodology that has a firm footing.

With respect to system specifications we must develop a specification language sufficient to capture all the requirements formulated above. This dictates that research must move away from considering specification languages that are limited to properties of information flow on noise-free channels to specification languages that can address, at the very least, general information flow, integrity, and availability. We must also move away from specification languages that are binary, in the sense that a system is described either as being secure or nonsecure, to languages that allow us to specify arbitrary sets of requirements that guarantee varying degrees of security. This does not mean that we should not continue work that is designed, for example, to capture "perfect confidentiality" [7,13], but that this work should be extended to "perfect security" and to allow for graceful degradation of these properties.

To guarantee that specifications are correctly implemented, we advocate the development of a system refinement methodology. Such a methodology will replace the current practice of building a system and then showing that it meets its specification by one where a system is developed in such a way that its specification must be met. This assumes that the specification addresses all concerns we are interested in and does not leave, for example, covert channel detection, until the stage when code is written. The work described in [14] can serve as a starting point since it shows how once a specification is proven to satisfy certain security properties, security concerns can be ignored during code refinement/verification. Since the only concern becomes functional correctness, the security community can borrow at will from the computer science community at large.

To make such a paradigm cost effective, we must develop a set of component specifications and a logic for reasoning about them. These specifications will be interface specifications and the logic will allow us to reason about composite systems made up from various components. To simplify the logic, the language used to specify the components should be the same as the language used to specify system requirements. We require that the logic be sound and that any verification system used to support it be highly assured. The system described in [14] can serve as a basis for this work since it provides a single sound and complete axiomatic system for reasoning about both specifications and programs and for reasoning about composability [15].

Summarizing these considerations, we arrive at the following proposals:

- Institute a research initiative in the area of trust analysis to determine the resources a penetrator is likely to expend to compromise various types of information. For any particular type of information, the resources one would expend to learn the information may differ from the resources one would spend to deprive legitimate users of the information and resources one would expend to alter the information.

- Institute a research initiative whose objective is to discover sets of system properties that will raise the cost of successfully compromising system security above those values determined in the above initiative and methods to show the appropriate-

ness of these properties. Properties required by perfectly secure systems (systems that allow no information flow over any channels, maintain perfect integrity, and are always available) should not be ignored, but rather used as a starting point from which other properties can be formulated by "graceful degradation." To this end we suggest using the work described in [7,13] as a starting point.

- Modify current specification and verification efforts to address the properties discovered in the above initiative. The focus here should be the development of refinement methods that yield correct systems rather than the analysis of systems after their development. The verification systems developed should be highly assured. We suggest using the work described in [14] as a starting point.

- Develop a set of components that can be used to implement the systems we desire and a verification method for reasoning about properties of composite systems made up of these components. It would be desirable if the composition logic resembled the refinement logic of the previous bullet. For this reason we suggest using the work described in [15] as a starting point.

5 Concluding Remarks

We have suggested initiating research in four areas: trust analysis, system property and specification development, refinement methodology, and composability logic. It should be pointed out that these are high risk efforts. For example, although recent espionage cases have shown that the replacement cost of highly classified information may not be as impossible to compute as some have assumed, we must also take into account the cost that may result from the loss of prestige that can follow information theft. Similarly, as recent debates in the cryptology community have shown, it is unclear how much various properties affect the cost system penetration. The outlook for security preserving refinement methods and composability logics are not certain either. Nevertheless, I do not see any alternative to initiating this research. We cannot afford to continue spending so much money on systems yet be so unsure about what our money has bought us in terms of protection. Better to face the risk of a lifeboat than to stay on a sinking ship.

Acknowledgements

I formulated many of the ideas in this paper while serving as Chair of the National Security Agency's Technical Exchange Working Group on High Assurance Software. Other members of the group were David Czaplicki of NSA, John Faust of Rome Laboratories, Judy Froscher of NRL, Jim Harper of NSA, Bobby Huynh of CECOM, and Carol Taylor of NSA. The paper has benefited from participant discussion during its presentation at the New Security Paradigms Workshop (September 22–24, 1992) and from audience discussion after its invited presentation at the National Computer Security Conference (October 13–16, 1992).

References

[1] W. Bartussek and D. L. Parnas, "Using Traces To Write Abstract Specifications For Software Modules," Report TR 77-012, University of North Carolina, Chapel Hill, N.C. (December 1977).

[2] D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, MITRE Corp., Bedford, MA (March, 1976). Available as NTIS AD A023 588.

[3] B. Boehm, "Software Engineering," *IEEE Transactions on Computers*, Vol. C-25(12) pp. 1226–41 (December 1976).

[4] European Communities Commission, "International Technology Security Evaluation Criteria," ISBN 92-826-3004-8, National Computer Security Center, Luxembourg (1991).

[5] Department of Defense, "Trusted Computer System Evaluation Criteria," CSC-STD-001-83, National Computer Security Center, Ft. Meade, MD (Aug. 1983).

[6] J. A. Goguen and J. Meseguer, "Security Policies and Security Models," pp. 11–20 in *Proc. 1982 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press (April, 1982).

[7] J. Gray, "Toward a Mathematical Foundation for Information Flow Security," *Proc. 1991 IEEE Symposium on Security and Privacy*, pp. 21–34, IEEE Computer Society Press, Oakland, CA. (1991).

[8] J. Gray and P. Syverson, "A Logical Approach to Multilevel Security of Probabilistic Systems," *Proc. 1992 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA. (1992).

[9] J. Jacob, "On the Derivation of Secure Components," *Proc. 1989 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA. (1989).

[10] D. McCullough, "Specifications for Multi-Level Security and a Hook-up Property," in *Proc. 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press (April 1987).

[11] J. McLean, "A Formal Method for the Abstract Specification of Software," *J. ACM*, Vol. 31(3) pp. 600–627 (July 1984).

[12] J. McLean, "Specifying and Modeling Computer Security," *IEEE Computer*, Vol. 23(1) pp. 9–16 (January 1990).

[13] J. McLean, "Security Models and Information Flow," in *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press (May 1990).

[14] J. McLean, "Proving Noninterference and Functional Correctness Using Traces," *Journal of Computer Security*, Vol. 1(1) pp. 37–57 (1992).

[15] C. Meadows, "Using Traces of Procedure Calls to Reason About Composability," *Proc. 1992 IEEE Symposium Research in Security and Privacy*, IEEE Computer Society Press, Oakland, CA. (1992).

[16] I. Moskowitz, "Variable Noise Effects Upon a Simple Timing Channel," *Proc. 1991 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press , Oakland, CA. (1991).

[17] D. Sutherland, "A Model of Information," in *Proc. of the 9th National Computer Security Conference*, Gaithersburg, MD. (September, 1986).