A Shift in Security Modeling Paradigms

James G. Williams
The MITRE Corporation
202 Burlington Road
Bedford, Massachusetts 01730

Abstract

Models of the external system interface of a computer have been successfully used to describe confidentiality requirements. This paper discusses the use of an external-interface model that supports the external consistency objective of Clark and Wilson as well as internal structural constraints needed to meet identified external-interface requirements. These internal constraints identify a vendor-supplied "Integrity Trusted Computing Base" that handles informal proofs called "pedigrees." The increasing use of external-interface models, which this work illustrates, represents a paradigm shift in the construction of security models.

1 Introduction

The paradigm shift referred to in the title of this position paper was first reported at the 1991 Franconia Computer Security Workshop IV [1] and again more recently in [2]. Briefly, the shift is from models that attempt to define security requirements in terms of controlled system entities towards more comprehensive models that begin with what are essentially "black box" requirements on the system interface. In the case of mandatory nondisclosure requirements, this shift was motivated by persistent reservations about the adequacy of traditional access-control models [3] and has led to newer models that include both noninterference-like requirements and more traditional internal constraints [4, 5]. These newer models show that, at the least, noninterference-like conditions can rule out a large variety of known covert channels. Moreover, to bring traditional access control models up to a comparable level of stringency, it is necessary to make some significant additions, including a variant of tranquility, definitions of what constitutes reading and writing, I/O constraints analogous to simple security and the star property, and a separate model of process scheduling [5].

A taxonomy of stages of elaboration in the development of trusted systems was presented in [1] whose first three stages may be summarized as follows:

- *Trust objectives* describe what is to be achieved by an information-processing enterprise, an important component of which is a computing system.
- *External-interface requirements models* describe the behaviors of computing systems, their users, and other entities in the systems' environments in such a way as to allocate responsibilities for achieving the identified trust objectives.
- *Internal requirements models* describe, in an abstract manner, how the system responsibilities given in the external-interface models are met within the system.

The primary purpose of [1] was to study the work of Clark and Wilson [6] through the use of external-interface modeling to see if a similar gain in clarity would result. The initial result of this effort was the identification of several system-interface requirements, some ofJwhich were implicit in the work of Clark and Wilson while others appeared to support the Clark-Wilson objectives but were not contained in their famous work.

After the 1991 Franconia Workshop, we selected external-consistency as the most prominent of the various Clark-Wilson objectives and began the process of elaborating this objective to obtain user responsibilities, external-interface requirements, and internal system requirements.

The following sections discuss the external-consistency objective, several supporting external-

*This paper reports on joint work with Leonard J. La Padula. This work was supported in part by the National Security Agency under MITRE Project 8350.

1993 ACM 0-89791-635-2

interface requirements, and some consequences for the internal structure of a computing system.

2 Formulating the External Consistency Objective

External consistency is the ability of a computing system to give correct information about its external environment. We begin with some typical examples. If an inventory program says that a warehouse contains given levels of various supplies, then the named supplies really can be found at the warehouse. If a bank statement lists a particular balance for an account, then the balance is correct as of the time it was issued. If a computing system labels an output as "secret," then the contained information has a classification level dominated by "secret." If a computing system issues a "sell" order for one million shares of stock, then that is the intent of its controlling organization. A "sunrise" program correctly predicts sunrises. This last, somewhat atypical example, is useful because of its stark simplicity.

To model external consistency, we need to account for the fact that computers are capable of producing output which users interpret as assertions about real-world entities (including, as a special case, the visible behavior of the computing system itself). In working formally with such assertions, there are several difficulties that do not arise in the propositions of first-order logic.

The truth or falsity of a real-world assertion can vary with time unless it contains explicit qualifications explaining when it is true. Another potential difficulty with real-world assertions is that they are only approximately correct unless level of accuracy is included in the assertion itself or in its context of interpretation. For example, the following assertion is very unlikely to be exactly correct because most events do not happen on minute boundaries:

system: *The sun will rise at 5:47 a.m. today.*

A third potential difficulty is that real-world assertions tend to rely heavily on context for their meaning through the use of complex semantic conventions.

For the sake of a simple model, we view a computing system through a stabilizing filter that maps each assertion input to or received from the system to a corresponding "stable" assertion whose truth is time-independent and whose accuracy is specified within the assertion itself. Thus, for modeling purposes, the above assertion might be replaced with the following:

system: *At Logan Airport on 23 September 1992, the sun rises between 5:46:59 a.m. and 5:47:01 a.m.*

In this form, the truth of the assertion depends only on the language used to express it (English) and on facts of astronomy but not on contextual information such as the time or place it was issued or the intent of its author.

Taking the above examples and issues into account, we are led to the following security objective:

2.1 External-Consistency Objective

Each assertion received from the system (and recast in stable form) is a true description of reality.

This objective applies uniformly to all assertions made by the system, including account status reports, financial transactions, purported facts about the behavior of programs, and so forth.

3 Requirements at the System Interface

According to the taxonomy described in [1], the second stage, after identifying basic objectives, is to allocate responsibilities to the system and its users. This allocation should be done in such a way as to ensure effective support even though users make mistakes and, in some cases, are maliciously motivated. After briefly mentioning user responsibilities, we consider some examples of system responsibilities, informal presentations of external-interface requirements, and, finally, what would be involved in a precise model.

The users must agree on a common language for describing real-world or other situations, and this language must contain a vendor-supplied sublanguage that is understood by the computing system so that the system can meaningfully participate in supporting the external-consistency objective. The assertions and requests which users input to the system must be correct; failing this, users must adequately warn the system of possible errors.

In talking about system responsibilities, we really mean vendor responsibilities deriving from vendor advertisements about support for external consistency. We want to discuss what the vendor-supplied hardware/software configuration should do to support external consistency. Consequently, we are interested in properties which are satisfied if the system is installed properly and has not been inappropriately tampered with.

There are surprisingly few assertions that a properly installed system might take full responsibility for.

Only trivial examples such as the following come to mind: when the computer is turned on, it mentions that the system clock has gained three days since it was last turned off. Let us see what a vendor might be able to say in support of the examples we have considered so far.

The simplest typical example is, perhaps, that of a nondisclosure label. The accompanying security documentation might explain that an output label provides a security level which dominates the information displayed, subject to various caveats: users have correctly identified levels of inputs on which the outputs were based, and there might be problems relating to covert channels, inference, and/or aggregation. A "Trusted Facility Manual" might further explain how to identify the users who contributed the potentially mislabeled information on which the output was based. Notice that external consistency goes far beyond the traditional notion of label integrity as discussed in the *Trusted Network Interpretation of the TCSEC* (TNI) [7].

In all of the remaining examples, the correctness of the explicit or inferred output assertion depends on the correctness of both application software and of user-supplied inputs. Even the sunrise program depends not only on the correctness of its software but also on the correct setting of the system clock. In all of these cases, the system can support external consistency by giving useful information about who supplied the inputs on which a given output depends. It can do this, in particular, for assertions about software correctness.

With sufficient care, it is possible for the system to fully comprehend the actions upon which correctness of output depends, to distinguish between relevant user actions and its own actions, to take responsibility for its actions, and to provide feedback on user actions that are crucial to the correctness of a given output.

3.1 Warranties on Correctness of System Output

In view of the above examples, the main external-interface requirement we shall impose in support of the external-consistency objective is the following:

Output-Warranty Requirement

> The system shall have the ability to mark some of its outputs as "warranted," meaning that these outputs are correct provided the inputs on which they are ultimately based are themselves correct.

We refer to the set of previous I/O events that the correctness of an output depends on as its I/O basis. The I/O basis for an output is allowed to include previous outputs as a matter of convenience (e.g., the basis for this month's bank statement includes the closing balance from last month's bank balance, which is appropriate, if it was not contested.)

The above output-warranty requirement is closely related to the following availability requirement, which we have not carefully studied: the system must be able to provide useful descriptions of I/O bases; in particular, these descriptions must not be too complex. In support of this availability requirement, the system typically identifies the user and/or user group responsible for each assertion in a basis. But the correctness of this identification relies on the Identification and Authentication (I & A) process as well as on the correctness of administrative information supplied by the system's security personnel. Usually, users are interested in "reduced" bases from which uninteresting caveats have been stripped (e.g., correctness of the I & A mechanism, correctness of administrative records, correctness of the system clock, etc.).

The above output-warranty requirement can be strengthened considerably in order to accommodate the possibility of incorrect inputs. One can require that outputs be based only on certifiably correct inputs and that the system be able to discard even certified inputs if they are later found to be incorrect. There are many possible forms of input certification. A common form is corroboration, in which a first input must be endorsed by a second input from a different, properly authorized user. The system's ability to recognize corroborated assertions may rest partly on administrative procedures which guarantee, for example, that no user has more than one login name.

3.2 An Error-Handling Requirement

So far, we have dealt only with "positive" information. We now talk about what to do when it becomes evident that a previously believed I/O assertion is incorrect. We want to selectively invalidate any other output assertions whose truth depends on this one. The word "selectively" is important here. Consider, for example, an input event which installs a program that has a virus in it. We might be able to discredit this program by erasing the system's disks and reprogramming the system from scratch, but we would prefer an easier, more selective approach, namely, that of just telling the system to disregard the infected program and all information that has been derived from it.

Error Suppression Requirement

> For any I/O assertion e, it is always possible to provide a later "invalidating" input e', such that, after e' has been input and processed, e will never again be used in the I/O basis of any later output.

This requirement does not actually say that an invalidating input must be selective. Selectivity is actually an availability requirement rather than an integrity requirement, but we mention selectivity in order to ward off trivial solutions to the error-suppression requirement.

4 Internal-Requirements Model

In order for a system to perform selective invalidation of information, it is necessary for it to track how each output and intermediate result has been derived. This derived requirement is essentially the *data continuity* requirement found in DOD Directive 5200.28J[8] (parent document to the TCSEC).

Fulfillment of the data-continuity requirement amounts to keeping a complete register of the events that have led to a given warranted result. This register, which we refer to as a *pedigree*, amounts to an informal proof of the correctness of the result relative to its I/O basis. When one looks closely, one discovers that these informal proofs rely not only on assertions in an I/O basis but on *postulates*, much as traditional proofs rely on axioms. In this case, postulates are assertions whose truth is established through direct observation and processing by trusted software. Thus, there is a notion of *Integrity Trusted Computing Base (ITCB)* which produces these postulates.

To provide an internal-requirements model that guarantees the above external-interface requirements, it is necessary to describe an interface language and to explain how it is used by the ITCB. One of the objectives of the internal-requirements model is to allow the construction of assertions that are based entirely on corroborated inputs. Such assertions have pedigrees, all of whose input assertions are corroborated. Another objective is to provide an interface language strong enough to support the kinds of warranties made by vendors about their own software.

The fundamental elements of the internal system policy are user ids, data items, data aggregates, pedigrees, and certified processes (CPs). Data items are analogous to the "controlled data items" of Clark and Wilson. Some data items are warranted assertions; some are "configuration items" of the sort found in configuration-management policies. Some warranted assertions are role authorizations. Some role authorizations allow certain users to function as security administrators. Each user group is defined by a set of role authorizations.

Certified processes are built up from certified state transformations; these may be classified as Integrity-Validation Procedures (IVPs) or Transform Procedures (TPs), as in Clark and Wilson, according to whether or not they explicitly manipulate basis descriptions.

While there are many details, the basic requirements of the internal model are easy enough to state. The main requirement is that, in every reachable state, every warranted assertion is supported by a valid pedigree. The notion of a valid pedigree is defined in terms of a state-dependent set of postulates. State-transition constraints are needed to explain how the ITCB adds new postulates.

The most sophisticated examples of error handling are perhaps associated with configuration management (CM) systems [9]. In such a system, the claim that a program serves a particular purpose is supported by what is essentially an elaborate pedigree that intertwines authorized inputs from system developers with assumptions about the behavior of their software-development tools. CM systems allow developers to maintain useful claims about software systems in the face of discovered errors and changing requirements, and their change-control mechanisms provide a useful starting point in designing specific mechanisms with which to satisfy the above error-suppression requirement.

5 Preliminary Conclusions

Strong support for the external-consistency objective entails several external-interface requirements, among them a requirement for vendors to produce limited warranties as to the correctness of outputs and a requirement to suppress erroneous information when errors are discovered and reported to the system.

Successful implementation of these requirements involves the ability to formulate "stable" assertions whose truth does not vary with the state of the system and to maintain registries or "pedigrees" that justify these assertions relative to an assumed I/O basis. Pedigrees must rely not only on user-supplied inputs but also on direct observations and other "postulates" whose correctness is the responsibility of an ITCB.

Using the mechanisms of an internal model, it should be possible to design an I & A mechanism that can produce pedigrees for assertions of the form "user U asserted A" in which all user assertions are corroborated. The necessary I & A requirements are much stronger than those of theJTCSEC but not much stronger than those of the draft "Minimum Security Functional Requirements" (MSFR) produced recently by the Federal INFOSEC Criteria Working Group. The main addition to the MSFR requirements is that, when a new user is introduced to the system, his identity must be corroborated by two system administrators.

The internal model provides a natural framework for discussing configuration management (CM), and traditional CM ideas provide a natural starting point for the construction of pedigrees and basis descriptions for certified procedures. Such pedigrees rest on certified assertions about the behavior of compilers, linkers, and configuration builders such as the UNIX "make" program. These tools are examples of TPs.

One can show that it is not possible to construct reliable pedigrees for CPs without access mechanisms stronger than those found in UNIX. Consider dynamic linking, for example. To conclude that a dynamically linked program satisfies a known specification, it is necessary to know that its linked subroutines behave as expected. In recognition of this, the UNIX ld.so linker checks version numbers. But its checks only work if the semantics of version numbers is respected. That is, (a) programs are not modified without incrementing version numbers, and (b) if only the minor version number is changed, then the new program is certified to satisfy the specification for the old program. Unfortunately, UNIX does not directly support the enforcement of these constraints.

References

[1] LaPadula, L. J., and J. G. Williams, June 1991, "Toward a Universal Integrity Model," *Proceedings of The Computer Security Foundations Workshop IV*, pp. 216–218, Franconia, NH.

[2] LaPadula, L. J., 13 January 1992, "Taxonomy for Stages of Elaboration of Requirements," Nuance Forum Entry LJL-3, Dockmaster.

[3] McLean, J., April 1987, "Reasoning About Security Models," *Proceedings of the 1987 Symposium on Security and Privacy*, pp. 123–131, IEEE, Oakland, CA.

[4] Gove, R. A., September 1984, "Extending the Bell & La Padula Security Model," *Proceedings of the 7th DOD/NBS Computer Security Conference*, NBS/NCSC, pp. 112–119, Gaithersburg, MD.

[5] Williams, J. G., May 1991, "Modeling Nondisclosure in Terms of the Subject-Instruction Stream," *Proceedings of the 1991 Symposium on Research in Security and Privacy*, IEEE, pp.J64–77, Oakland, CA.

[6] Clark, D. D., and D. R. Wilson, April 1987, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 Symposium on Security and Privacy*, IEEE, pp. 184–194, Oakland, CA.

[7] National Computer Security Center, July 1987, Trusted Network Interpretation of the TCSEC (TNI), NCSC-TG-005, National Computer Security Center, Linthicum, MD.

[8] Department of Defense, March 1988, Security Requirements for Automated Information Systems (AISs), DOD Directive 5200.28.

[9] Witgift, D., 1991, *Methods and Tools for Software Configuration Management*, John Wiley & Sons.