COMPOSABLE TRUSTED SYSTEMS

E. S. Lee
P. I. P. Boulton
B. W. Thomson
R. E. Soper

Technical Report CSRI-272
May 31, 1992

Computer Systems Research Institute
University of Toronto
Toronto, Ontario, Canada M5S 1A4

The Computer Systems Research Institute (CSRI) is an interdisciplinary institute formed to conduct research and development relevant to computer systems and their applications. It is supported in part by the Natural Sciences and Engineering Research council of Canada.

This document is the final report of Contract W2213-9-6351/01-SZ issued by the Department of Supply and Services of the Government of Canada. The funding was administered by the Communications Security Establishment of Canada. Milan S. Kuchta was the scientific authority for the contract.

[Wittbold90] Wittbold, J.T., and D.M. Johnson, ''Information Flow in Nondeterministic Systems'', *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, Ca., pp. 144-161

[Yu88] Yu, C.-F., and V.D. Gligor, ''A formal specification and verification method for the prevention of denial of service'', *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE 1988, pp. 187-202.

Throw this page away.

Throw this page away.

Throw this page away.

TABLE OF CONTENTS

PREFACE

The ability to derive the properties of an assembly of components from the properties of its individual constituents is central to good engineering design practice. It is also an underlying axiom in the modern discipline of computer system engineering. A system for which this capability exists is said to be a *composable* system.

In the design of a system, it is desirable to start at quite a high level of abstraction with the overall functional requirements of the system and to step-by-step work towards the details of the individual sub-systems that will be needed. Composability is essential for this successive-refinement design methodology.

Classical successive-refinement design methodologies begin with the creation of an abstract, high-level specification of the intended functional behaviour of a system. This specification is intentionally vague about non-critical details, so that constraints are applied only where required by overall system goals. Step-by-step refinement is applied to the abstract specification to evolve it into a suitable implementation. Details that were not visible or not fully constrained at the abstract level are determined during this process. More and more details of the system behaviour are determined at each step of this refinement process.

The design process starts at the highest level of abstraction, with a full system view. The design is then successively refined to determine the specifications of the components of the system. The system will then meet its overall requirements because each component is designed specifically to do just that.

For many years engineers have been designing systems using standard, pre-designed components to achieve an economy of design effort and multiple other benefits. The system is composed out of sub-systems that have been designed individually and separately. This means that the designer need not start with a blank sheet every time a system is to be composed, because a repetoire of components exist that can be incorporated into a new system.

What is required is that these sub-systems be composable. This means that as long as some simple rules are followed, they can be connected to each other, or to purpose-designed interface components, and the result will be a system that meets some overall criteria determined by the composability properties of the components.

Composability is a desirable goal. We are interested in composing secure information processing systems out of secure components that have been shown to be composable. The individual parts have been evaluated and found to be composable in certain ways. If the proper rules are known to have been followed, and if any purpose-built components that have been included in critical sections of the system have been evaluated and shown to be composable, the system must be secure.

The thrust of this project was to perform research to attempt to find ways to build, categorize, describe, and assemble composable components of secure information processing systems. The successive-refinement methodology may be at odds with the preservation of security. If we think of constructing a secure system out of a collection of parts, each of which has been independently evaluated to meet some security requirements, we can easily arrive at a resultant system that is not secure.

In order to deal with the issue of the level of confidence that can be placed in a purportedly secure system or sub-system, it must possible to measure the assurance that the system or sub-system meets the requirements of a set of criteria. To develop such a measure, there must be some set of values that the abstract notion of assurance can assume, and some way to gauge or quantify the assurance of the system. We want to be able

- to define the security policy that a system must meet, and then
- to derive the functionality that is required of the system, and then
- to develop a measure of the assurance that we require that these goals are met.

We want to do these things without confusing the notion of assurance with the notion that we think of as a guarantee. Additional mechanism may be necessary to provide the function demanded by a security policy, and thus to guarantee that it is fulfilled. Assurance should be a measure that is independent of this, a measure of the level of confidence in the validity of the guarantee.

A guarantee asserts that given security functionality is sufficient to satisfy the requirements of a security policy. Usually, there will be levels of guarantee that can be given, corresponding to levels of functionality provided by the system.

Assurance is associated with any guarantee. It is the level of confidence that a guarantee of security is valid and effective. It is associated with both the quality and the durability of the guarantee.

Suppose that some system is to provide appropriate user functions, and that it must satisfy some known security policy. Some security functions will be implied by the security policy. It must be the case that suitable mechanisms will exist in the system to provide these security functions. Thus, since it provides the requisite function, the overall system gives a guarantee that the security policy is obeyed. The details of what specific functions must be guaranteed to be present, or to be absent, is a significant aspect of the specification of the system. The guarantee must address the intended use or possible misuse of the system, with the risks of security failure and its attendant damage considered. In practice there would undoubtedly be standard guarantees that would suffice for the majority of systems that could be implemented.

The assurance associated with the guarantee, and thence with the system, is a measure of the confidence that the system will enforce the policy without exception or qualification. The assurance is the level of trust that might be placed in the guarantee. The assurance will be determined from many factors.

This research delves into the notions of guarantee and assurance of composable systems. Thinking clearly about the nature of guarantee and assurance is necessary to study the composition of systems. Composing two systems that have a different guarantee is considerably different than composing two systems with a different assurance. We will show that drawing the distinction between guarantee and assurance will help the composition problem.

This is the final report on work performed on DSS Contract W2213-9-6351/01-SZ for the Communications Security Establishment of the Department of National Defence, Ottawa. Much of work under this contract has been reported previously in a series of four-monthly progress reports beginning October 16, 1990. This report includes the results from the progress reports, so that it can be read without reference to them. The work that is included was performed by a team of four research workers. They are:

E. Stewart Lee	Principal Investigator
Peter I.P. Boulton	
Robert E. Soper	
Brian W. Thomson	

Brian Thomson and Bob Soper did most of the day-to-day research. Much of what is of value in the results described in this report is directly attributable to one or both of them. Stewart Lee was responsible for defining the job to be done, for many of the original ideas behind the work, and for the management of the contract that resulted. Peter Boulton participated with the others in the research, in the contract management, and particularly in preparing this final report.

E. Stewart Lee
Toronto, May 31, 1992

1 INTRODUCTION

A little monograph on
the ashes of one hundred and forty
different varieties of pipe, cigar, and cigarette tobacco.
Sir Aurther Conan Doyle,
The Adventures of Sherlock Holmes,
The Boscombe Valley Mystery
1891.

To see so far as one may,
and to feel the great forces that are behind every detail ...
to hammer out as compact and solid a piece of work as one can,
to try to make it first rate ...
Oliver Wendell Holmes, Jr.,
The Class of '61. From Speeches,
1913.

1.1. INTRODUCTION

This document reports on our research to develop measures and means for the composition of secure computer systems from components known to be secure. It is a fundamental principle of this work that the security issues that affect the trustworthiness of subsystems and of composite systems may be related to *guarantee* , which refers to properties of the system specification, and related to *assurance* , which refers our level of confidence that the implemented system conforms to the guaranteed specification. Consequently, the scope of this work includes the composition of guarantee - that is, how to define the functionality of components so that their composition retains desired (security) characteristics - and the composition of assurance - appropriate measures for assurance and techniques for combining these measures in a composite system.

1.1.1. Guarantee and Assurance

In order to deal with the issue of composition of guarantee and assurance, we must first categorize security concerns as being related either to one or the other. In distinguishing the two categories we need to apply a consistent set of definitions. The statement of work issued for this research contained definitions of these terms, and required investigation of the concepts embodied in the definitions. A guarantee was defined as

> an assertion that a given security functionality is sufficient to satisfy the requirements of a specified security policy

while assurance was

> the level of confidence that a guarantee associated with a specified security policy is both valid and effective and that assurance is associated with both the quality and durability of the guarantee.

In these definitions, the term *security policy* is considered to designate

> the set of laws, rules, and practices defining the protection of a system and the information it contains from unauthorized disclosure, modification, destruction, or removal and protection from the interruption of services that the system provides.

Our objectives therefore include developing methods of measuring guarantee and assurance, and methods for deriving the guarantee and assurance of a composed system from the guarantees and assurances of its components.

1.1.2. National Security Criteria

In approaching this research, we believed it important to relate our work both to the formalism of computer science and the practicalities of the published national security criteria documents of several countries. It was our hope that the latter effort would provide guidance for our modelling effort in the form of a target based on official policy statements.

1.2. OVERVIEW

Chapter 2 sets the stage for our work by providing a review and summary of the national computer security criteria published by the United States of America and Germany, and as expressed in the European harmonized security criteria.

In chapter 3, we begin our examination of guarantee and assurance. Guarantee must be composed first, since an assurance without a guarantee is meaningless. Therefore, chapter 3 concentrates on the composition of specifications, and leaves until later the issue of conformity with specification. This is equivalent to making the assumption that all components are utterly reliable in the sense that their behaviour is fully in agreement with their specification (their ‘guarantee’), and then asking what extra security specifications must be applied to the components such that their composition will continue to behave appropriately.

Assurance is taken up in chapter 4. The published security criteria have little to say on the issue of assurance, so much of this chapter deals with the concept of assurance as expressed in other standards documents.

In chapter 5, we begin a survey of the formal models for security to be found in the literature of computer science and engineering. We concentrate on the much researched policies relating to confidentiality. We extend our critique of formal modelling to issues of integrity and denial of service in chapter 6.

Nondeterminism has been the spark for many new models and modelling methods since its introduction to the security scene in 1986. In chapter 7, we review this work and present and support our thesis that nondeterminism is neither necessary nor desirable, and that existing nondeterministic models fail to adequately address the security problems cited as evidence supporting the need for nondeterminism.

Our main results are presented in chapters 8 and 9. We propose that the problem in generating composable security properties results from the attempt to define security for isolated components, and subsequently retrofit composition into the model. We call these ‘‘open’’ models, in that they are based on the analysis of an individual component operating in isolation, without regard to what might actually be producing or consuming the signals that enter or exit it. This chapter suggests the use of ‘‘closed’’ modelling as a basis for the definition of security properties, where a closed model is one in which the component under scrutiny is assumed from the start to be interconnected with an assortment of information sources, sinks, and processors.

In chapter 9, we present a formal logical modelling method for security properties, and translate existing confidentiality models to it. We obtain the expected immediate result that in general, these confidentiality models are not composable. We conclude with a new flow model for confidentiality, the Possibilistic Flow Model. This is a closed model of strong confidentiality with the non-probabilistic simplicity of nondeterminism, but the composability of flow models.

2 GUARANTEE IN NATIONAL COMPUTER SECURITY POLICIES

My history has been composed
to be an everlasting possession,
not the showpiece of an hour.
Thucydides,
The History of the Peloponnesian War,
431-413 BC.,
Book I, section 22.

Guarantees which are not worth the paper they are written on.
Johann Bernhard,
Graf von Rechberg,
In a dispatch concerning the recognition of Italy,
1861.

2.1. INTRODUCTION

Defining a ''measure'' of guarantee is an elusive task. Just what is guarantee, and how can it be quantified? We must develop a quantitative notion of guarantee if we expect to compose trusted systems and deduce the resulting composite guarantee.

Since guarantee is fundamentally a measure of a system's security functionality, we examine here some widely-held notions of trustworthiness assessment. Later in this report we will survey the various notions of guarantee and guarantee measure suggested in the literature by formal security modellers. In this chapter, we direct our attention to another set of documents that indicate the perspectives of those who set security policy. These are the ''national computer security policies'' that various governments are promoting as standards for evaluating trusted systems. We hope to gain from these considerations a set of principles which together make up our intuitive notion of guarantee, and each of which, individually, is relatively straightforward to measure.

In the United States, there exists the *Trusted Computer Systems Evaluation Criteria* [TCSEC85], also known by the acronym TCSEC or, more colloquially, as the ''Orange Book'', that establishes a graded set of measures against which a computer system can be calibrated to establish levels of security confidence. We first explore the hierarchy of security guarantee as it is embedded in that document.

2.2. THE ORANGE BOOK

The Orange Book divides the fundamental computer security requirements into three groups: Security Policy, Accountability, and Assurance. The detailed criteria statements further subdivide these into nine requirements:

Security Policy:
- Discretionary Access Control
- Object Reuse
- Mandatory Access Control
- Labeling

Accountability:
- Identification and Authentication
- Audit

Assurance:
- Operational Assurance
- Life-Cycle Assurance
- Documentation

Neglecting the Assurance requirements for the time being, we will examine the detailed specifications contained in the Orange Book relating to the Security Policy and Accountability requirements.

The Orange Book contains specifications for six classes of trusted systems. We will deal only with the first five since the last two differ only in their Assurance requirements. These five classes (C1, C2, B1, B2, and B3) each contain different combinations of specifications for the six Security Policy and Accountability requirements, ordered such that each successive class contains all the security mechanisms of its predecessors.

Thus the classes form an ordered progression from least-trustworthy (C1) to most-trustworthy (B3), but it is interesting that, given six rather broad categories of requirements, only five levels of trustworthiness were defined (again, neglecting the Assurance requirements). As we will see, this is partly because some aspects of the different requirements are highly interdependent, and partly due to a desire to create a small number of trust classes. This desire may spring from the difficulty in making fine distinctions during the accreditation process, whereas we are interested in making the finest distinctions possible.

2.2.1. Discretionary Access Control

The Orange Book defines Discretionary Access Control (DAC) as:

> A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

Discretionary Access Control allows access to an object to be controlled by those subjects who already have access to it. This is in contrast to Mandatory Access Control, which controls access to objects based on global notions of sensitivity and clearance.

Also, the Orange Book requires only that Discretionary Access Control be enforced on ''named objects'', in contrast to the requirement that Mandatory Access Control apply to all objects, whether or not a subject can refer to them by name. We will make note of this again later.

Three levels of Discretionary Access Control are specified in the Orange Book. There are minimal requirements at class C1, additional requirements at C2, and further requirements at B3. It should be noted that all the requirements stated in the Orange Book are minimal, in the sense that conforming systems can enforce stronger requirements as long as the specified criteria are satisfied.

2.2.1.1. C1 Discretionary Access Control

At class C1, Discretionary Access Control need not apply to all named objects in the system. Thus, some objects may be subject to unlimited access, but the capability must exist for specific objects to be brought under Discretionary Access Control at the request of some user.

This access control, however, need only distinguish accesses by users in distinct named groups. Thus it might be impossible for a user to allow only himself access to an object that he created, unless a group exists of which he is the only member. This may well not be the case since it is not required that users may define their own groups.

Additionally, groups need not be defined as a list of user names; in fact, individual users do not have to be uniquely identifiable. It is acceptable for users to identify themselves to the Discretionary Access Control mechanism as members of some group.

Also, at this level there is no distinction made between the types of accesses that subjects can make. The ability to read an object implies the ability to write to it, as well as the ability to delete it. Therefore, there is no notion of individual ''ownership'' in C1 systems. Objects belong to groups of users, or to all users. Since no one user controls a specific object, any user with access to an object can change its access groups as well. Thus it is possible for the creator of an object to be denied access to it by another user.

Clearly, C1 systems are intended for use in environments where users can be expected to cooperate, and where the main purpose of the security mechanisms is to prevent accidental damage.

2.2.1.2. C2 Discretionary Access Control

At class C2, Discretionary Access Control must protect all named objects in the system. In the event that a user does not specifically request that access be limited in some way, a default access restriction must be provided.

Access must also be controllable to the granularity of individual users, so that specific users can be allowed access if desired. If group access controls are provided, these groups must be defined as lists of unique user identifiers.

Most importantly, special authorization is needed to be able to change an object's access permissions. A ''control'' permission, distinct from access permission, must be defined per object, so that users with access to an object cannot, in general, change that object's accessibility. Those possessing this control permission can therefore be considered ''owners'' or administrators of the object.

2.2.1.3. B3 Discretionary Access Control

At class B3, access must be specifiable by listing individual users and groups of named users for whom access is allowed, as well as those users and groups for whom access is explicitly disallowed.

Also, different object access modes (e.g. read, write, delete) must be provided. There are no required access modes, but finer distinctions than access/no access must be achievable.

2.2.1.4. Guarantee

We can organize the Discretionary Access Control requirements into three independent considerations:

- which objects need access controls (domain),
- how finely to distinguish among users (control granularity), and
- how finely to distinguish access rights (access granularity).

Each level of Discretionary Access Control can now be seen as defining a location along each of these three axes. The following table summarizes the Orange Book Discretionary Access Control requirements:

	C1	C2	B3
Domain	some objects	all named objects	all named objects
Control Granularity	groups	individual users	users, excluded users, groups, excluded groups
Access Granularity	all/none	control/access/ both/none	control/...others.../ none

Clearly, many finer distinctions among Discretionary Access Control policies could be drawn. We will consider some general evaluation schemes for Discretionary Access Control after investigating the European approach.

2.2.2. Object Reuse

The Orange Book requirements for Object Reuse are quite simple. For classes C2 and above:

> No information produced by a prior subject's actions is to be available to any subject that obtains access to a (storage) object that has been released back to the system.

This statement effectively requires the elimination of all covert storage channels involving reusable shared system resources. It seems to be necessary because Discretionary Access Control only applies to named objects. ''Anonymous'' objects created in the course of a program's execution, say, are not subject to Discretionary Access Control and thus need protection from reuse by other subjects.

Mandatory Access Control, if interpreted in a reasonably broad sense, does protect the information in unnamed system objects, rendering the Object Reuse requirement somewhat redundant. The Object Reuse requirement is quite a bit stronger, however, since Mandatory Access Control would permit reading of ''leftover'' information by dominating subjects, and writing by dominated subjects.

Thus the Discretionary Access Control and Object Reuse requirements appear to form a closely-related pair, as do the Mandatory Access Control and Labeling requirements. The Discretionary Access Control and Object Reuse requirements protect named and unnamed objects, respectively. In fact, Object Reuse can be viewed as a highly-restrictive application of Discretionary Access Control: information in unnamed system objects can only be accessed by the creating subject.

2.2.2.1. Guarantee

Although the Orange Book treats Object Reuse as a single atomic requirement, we can envision quantifying it by considering:

- which reusable objects require purging of information,
- which subjects have access to unpurged released objects, and
- the effectiveness of the information purging.

The Orange Book requires that all released objects be purged before reuse, but, depending on the specific implementation, it may be quite practical (or perhaps unavoidable) for a system to allow some resources to be reused intact. Thus it seems reasonable for guarantee to increase as a larger fraction of a system's reusable objects is purged upon release.

Guarantee for the second aspect of Object Reuse increases as fewer subjects are allowed access to released objects, or as the privilege required for that access increases. For example, a system with very low guarantee in this aspect might allow every subject access to reused objects. Slightly higher in guarantee might be a system allowing access by subjects dominating (in the Mandatory Access Control sense) the information in the objects, as mentioned previously. This approach would likely be feasible and efficient on machines with hardware support for enforcing Mandatory Access Control on regions of memory.

Higher still in guarantee would be a scheme allowing subjects access only to information in reusable objects that they themselves had released. An even stronger requirement would allow only the system security officer access to released objects, presumably for some type of auditing purpose. Finally, the highest guarantee would be provided by allowing no subject, including the subject that released it immediately before, to access information in reusable objects. This is the level of guarantee provided in the Orange Book.

The third aspect of Object Reuse guarantee can be measured by examining the ''left-over'' information content accessible to subjects which are not allowed access. Thus, a very low guarantee results if the information in released objects is untouched before reuse. Higher guarantee would be obtained if the information is encrypted or otherwise randomized prior to reuse. The highest guarantee is provided by clearing all information completely from the released object, by setting all bits in its representation to zero, for example. This, again, is the level of guarantee provided in the Orange Book.

2.2.3. Mandatory Access Control

Mandatory Access Control (MAC) is defined in the Orange Book as:

> A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e. clearance) of subjects to access information of such sensitivity.

In contrast to Discretionary Access Control, this scheme protects objects based on their information content. Ordinary users cannot grant access to objects; some central authority certifies (in effect) all access requests.

The Orange Book specifies two levels of Mandatory Access Control. The basic Mandatory Access Control policy must be enforced at B1, and a strengthened requirement comes into effect at B2.

2.2.3.1. B1 Mandatory Access Control

At class B1, Mandatory Access Control must protect all storage objects under control of the Trusted Computing Base (TCB). Subjects and objects must be labeled using the well-known scheme of hierarchical levels and non-hierarchical categories, with at least two hierarchical levels available for labeling.

Labels serve as the basis for enforcing the Simple Security (‘‘no read up’’) Property and the * (‘‘no write down’’) Property, and for ensuring that all subjects created by a user have a clearance dominated by that of the user.

2.2.3.2. B2 Mandatory Access Control

Class B2 Mandatory Access Control differs from the B1 requirements in only one respect. The access controls must now extend to cover all resources directly or indirectly accessible by subjects. For systems on which the TCB controls all subjects and objects (i.e. there is no untrusted portion), this appears to imply only extending Mandatory Access Control to I/O devices. Information imported and exported by the system must already be labeled and protected in class B1.

2.2.3.3. Guarantee

We can organize the Mandatory Access Control requirements into three independent considerations:

- which objects need access controls (domain),
- how finely to distinguish levels (level granularity), and
- how finely to distinguish categories (category granularity).

Each level of Mandatory Access Control can now be seen as defining a location along each of these three axes. The following table summarizes the Orange Book Mandatory Access Control requirements:

	B1	B2
Domain	all storage objects	all resources
Level Granularity	> 1	> 1
Category Granularity	no requirement	no requirement

Again, many finer distinctions could be drawn.

2.2.4. Labeling

The Orange Book Labeling requirements are closely tied to Mandatory Access Control. These are mainly common-sense restrictions on the pairing of subjects and objects with their labels.

2.2.4.1. B1 Labeling

At class B1, all subjects and all storage objects must be labeled. This includes any data originating outside the system, which must be labeled by an authorized user.

Any information exported from the system must also be labeled. Each communication channel and I/O device is designated as single- or multi-level, and information passing over a multi-level channel must be unambiguously paired with its label. Any change in level of a channel or device must of course be controlled by the TCB.

2.2.4.2. B2 Labeling

At class B2, users must be immediately notified of any change in their clearance. They must also be able to query their clearance at any time.

Also, since B2 Mandatory Access Control extends to I/O devices, all devices must be assigned minimum and maximum security levels, reflecting the devices' ‘‘physical environments.’’

2.2.4.3. Guarantee

Very little of the Orange Book Labeling requirements specify more than the minimum functionality necessary for implementing Mandatory Access Control. However, there are aspects of Labeling itself that are sufficiently distinct from access control considerations that they may serve as useful metrics for Labeling guarantee. These relate to the ‘‘reliability’’ of labels:

- how well labels reflect the true level of their associated information, and
- how tightly labels are bound to their associated information.

The first consideration relates to a potential problem with any Mandatory Access Control implementation: who decides the level of information when it is created? It appears that a higher measure of guarantee should accrue when an authorized security officer makes this determination, rather than allowing the creating user to do so. In fact, one can imagine scenarios wherein groups of highly-trusted users determine the level of certain highly-classified pieces of information. This should imply a much higher measure of guarantee than if the creator(s) were alone given this responsibility.

In fact, this problem takes on a much greater importance when we begin to consider the composition of systems: how much should one system trust another system's labels? Assuming no errors occur in communication, this depends mainly on the quality of ''separation'' observed in the other system. This topic is discussed in some depth in chapter 4 of the final Trusted Network Architecture report [Thomson88].

The second consideration above is a measure of the ''strength'' of an implementation's labeling support. Mandatory Access Control increases in effectiveness as labels become progressively more intrinsic properties of the information stored in objects. For example, a high measure of guarantee should result when an object and its label are cryptographically bound together, so that they cannot be separated outside of the TCB. Likewise, hardware support such as restricting different levels of information to reside in distinct regions of memory, or using tag bits to label individual words, should lead to a high guarantee for this aspect. Alternatives to this type of low-level information labeling include formal verification of program source code and use of ''certified-trustworthy'' system programs.

2.2.5. Identification and Authentication

The Identification and Authentication requirements in the Orange Book exist to support the provision of discretionary and mandatory access controls. Thus the graduated requirements detailed below parallel the graduated Discretionary Access Control and Mandatory Access Control requirements.

2.2.5.1. C1 Identification and Authentication

At class C1, users must identify themselves to the TCB before accessing protected information. Some protected mechanism must be used for the authentication, and the authentication data must be protected from unauthorized access.

2.2.5.2. C2 Identification and Authentication

Since C2 Discretionary Access Control requires protection for all named objects, the additional C2 Identification and Authentication requirement is that all users must be uniquely identifiable.

2.2.5.3. B1 Identification and Authentication

Since Mandatory Access Control comes into effect at B1, the Identification and Authentication requirement at B1 is that the TCB maintain authentication and clearance information for each user.

2.2.5.4. B2 Identification and Authentication

At class B2, a trusted path is required for login. The establishment of this trusted path must be initiated by the user. This requirement seems in keeping with the extension of Mandatory Access Control at B2 to the protection of I/O devices.

2.2.5.5. B3 Identification and Authentication

Class B3 Identification and Authentication makes additional demands on the trusted path mechanism. A trusted path must be used whenever a ‘‘positive TCB-user connection’’ is required, not only at login. Additionally, this path must be ‘‘logically isolated and unmistakably distinguishable’’ from other paths.

2.2.5.6. Guarantee

The Orange Book Identification and Authentication requirements can be organized along the following three axes:

- what users need to be identified,
- what information must be retained per user, and
- the strength of authentication required.

Although C1 systems do not require every system user to be uniquely identified, this is clearly a necessity when a reasonable amount of trust is required. Also, the information maintained per user is mainly dependent on the nature of the access controls and authentication mechanisms in effect.

The last consideration is the central issue here. It is complicated by the wide variety of practical authentication schemes, and by the changing requirements on authentication schemes in different situations. Since this topic is addressed in some detail in the German criteria, we will reserve further comment until that has been discussed.

2.2.6. Audit

The requirements for Audit in the Orange Book are rather demanding at class C2, and grow in complexity at higher levels. Auditing essentially provides a trail of accesses to protected objects, hopefully useful in providing assurance of a system's secure status, and in tracing activities that may have led to security violations.

2.2.6.1. C2 Audit

At class C2, the audit trail consists of events of the following types:

- identification and authentication transactions,
- introduction of objects into a subject's address space,
- deletion of objects,
- actions of administrators, and
- other ''security relevant'' events.

Each event in the audit trail must contain:

- the event's type,
- the time at which it occurred,
- the user involved, and
- whether the event was successful (if applicable).

For identification and authentication events, the logical/geographical origin of the request must be included. For events concerning introduction and deletion of objects, the name of the object must also be included. Also, the facility must exist for particular users' actions to be audited selectively.

2.2.6.2. B1 Audit

Class B1 introduces additional requirements that are clear manifestations of the Mandatory Access Control requirements at this level.

First, an additional event type is required to audit any override of human-readable output labels. In addition, events concerning introduction and deletion of objects must also include the object's security level. Also, actions must be selectively auditable based on the security level of the object involved.

2.2.6.3. B2 Audit

At class B2, the only additional requirement is for the audit of events which are believed exploitable as components of a covert storage channel. This parallels the requirement that a ‘‘thorough search’’ for covert storage channels be conducted at B2, a task that forms part of the Assurance requirement of the Orange Book. This will be discussed later.

2.2.6.4. B3 Audit

The B3 audit requirements demand a sophisticated mechanism for analyzing events as they occur, in order to detect imminent violations of the security policy. Once some ‘‘thresholds’’ have been exceeded, the administrator must be notified, and, if necessary, some actions may have to be taken to actively prevent the violation from occurring.

2.2.6.5. Guarantee

We can organize the Orange Book Audit requirements into these four considerations:

- which events are audited,
- the completeness of the event report,
- the flexibility of audit control, and
- the extent to which the audit trail is automatically analyzed.

It seems clear that a system’s Audit guarantee should increase as more information is captured in the audit trail. However, measurement along this axis is complicated by the varying security-relevance of different types of auditable events. Some ordering among these events must be defined such that audit trails containing events of high relevance to security receive a higher guarantee rating than more voluminous audit trails containing events of low importance.

Likewise, the information recorded per event report must be measurable in a security-relevant manner for comparison of Audit guarantees to be meaningful.

Measuring flexibility of audit control is problematic at first glance, but certainly the control which a security officer wields over audit trail production is a very important measure of a system’s resistance to unauthorized use. We will see, in fact, that flexibility of control over a system’s security features is one of the major principles behind guarantee assessment.

Finally, the provision of tools for the automated analysis of audit trails vastly improves the utility of auditing, but again, comparing guarantees in this respect will be a difficult task.

2.2.7. Conclusions

We have produced, by a careful examination of the security functionality requirements of the Orange Book at each class, a set of principles which characterize a system's trustworthiness in a relatively straightforward manner. Each principle is largely independent of the others, and, we hope, spans the space of possible trusted systems which adhere to the underlying policies in the Orange Book.

We will return to discuss the ''independence'' and ''completeness'' of this set of principles, after considering the German and European approaches. At that time we will form some general principles of security assessment which should make possible the accurate determination of a system's guarantee.

2.3. THE GERMAN SECURITY CRITERIA

In 1989 the German Information Security Agency published "IT Security Criteria: Criteria for the Evaluation of Trustworthiness of Information Technology Systems"[ITSEC89]. This document contrasts sharply with the American TCSEC of 1985, and not only due to its green colour. The Green Book is less formal, less redundant, and less rigidly prescriptive than the Orange Book. It offers a more comprehensive treatment of computer security issues, but would likely be less useful than the Orange Book as a guide for evaluating contemporary trusted systems. However, for our purposes, the Green Book is an abundant source of information and ideas. We will study it in some detail.

It should be noted that the Green Book discussed here is labeled a "First Version", and was meant to be rapidly updated, as well as accompanied by more detailed manuals on threat assessment and trusted system evaluation. Moreover, it has likely been rendered obsolete by Germany's participation in the "Harmonized Criteria". However, the latter document has appeared only in a preliminary draft form, and follows the general form of the German effort, though it happens to eliminate much of the material that we will be examining below. Therefore, we feel that the Green Book is currently of great value, mainly for its enumeration and organization of contemporary security issues.

Fundamentally, the Green Book differs from its American predecessor by not advocating a particular security policy, but attempting to address the security concerns of a wide variety of commercial and governmental users.

A very broad spectrum of security policies can be addressed by dealing with the following eight "Security Functions":

- Identification and Authentication,
- Administration of Rights,
- Verification of Rights,
- Audit,

- Object Reuse,
- Error Recovery,
- Continuity of Service, and
- Data Communication Security.

This is an interesting classification scheme that contrasts with the Orange Book's main groupings:

- Discretionary Access Control,
- Object Reuse,
- Mandatory Access Control,
- Labeling,
- Identification and Authentication, and
- Audit.

The Green Book's "Rights" categories subsume the MAC and DAC groupings, and the "Error Recovery" and "Continuity of Service" functions address areas that the Orange Book considers assurance requirements. The Orange Book's "Labeling" considerations are dealt with by the Green Book's "Identification and Authentication" function. We will consider these contrasting perspectives when we deal with the Security Functions in turn below.

A particular security policy is defined by stating specific requirements for each security function. Conformance of an implementation to the policy is then evaluated by examining the mechanisms (e.g. algorithms and architectural features) employed for each function. Mechanisms are scrutinized to determine whether:

1. they are sufficient to completely enforce the security policy,
2. given correct implementation, any fundamental weaknesses exist that allow policy violation, and
3. the mechanisms have been correctly implemented.

These three considerations address aspects of both guarantee and assurance. Item 1 is determined by the quality of the mapping from security policy to system specification, and item 3 must be evaluated after implementation. Item 2, on the other hand, is a question of "design", and is therefore a different type of consideration. Although we have yet to rigorously define the terms *guarantee* and *assurance,* it seems clear that guarantee is fundamentally a measure of design quality, while assurance is a measure of the qualities of the design and implementation processes. Thus, we will concern ourselves here (as does the Green Book) with the evaluation of mechanisms to determine their effectiveness in the absence of implementation flaws (that is, with item 2).

The fundamental weaknesses of item 2 above are of two kinds:

- inherent weaknesses of the mechanism which cannot be practically eliminated by organizational means (that is, by means external to the computer system itself), and
- weaknesses which can be effectively reduced or eliminated by organizational means.

The rating of a mechanism is complicated by weaknesses of the latter kind, since it then involves evaluation of a much larger field of activity. It is essential, therefore, that it be ascertained for each mechanism employed:

- which aspects of the security policy does the mechanism attempt to address,
- what fundamental weaknesses does the mechanism possess (and their possible effects on other related mechanisms), and
- what organizational measures must be employed to ensure the mechanism's effectiveness.

Given these last two items, a measure for the mechanism can be ascertained. A six-level quantization of mechanism effectiveness is recommended:

Ineffective
: The mechanism has no (positive) effect on policy enforcement.

Weak
: The mechanism is suitable for preventing accidental policy violations, but offers little resistance to deliberate attempts at violation.

Moderate
: The mechanism provides some protection against deliberate violation, but can be defeated with moderate effort by those familiar with the system.

Strong
: The mechanism provides good protection against deliberate violation, and can be defeated only with great effort or extensive insider support. Any organizational measures necessary to support this level of effectiveness must be simple and reliable.

Very Strong
: The mechanism provides very good protection against deliberate violation, and can be defeated only with great effort and very extensive insider support. Any organizational measures necessary to support this level of effectiveness must be relatively simple and largely unsusceptible to fault or error. Monitoring and control of these measures must be integral to the mechanism and rated as a part of it.

Virtually Unbreakable
: The mechanism prevents all violations of the relevant aspects of the security policy, and is practically impossible to overcome given the current state of the art. All organizational measures employed must be completely protected from error by internal monitoring which forms an integral portion of the mechanism.

Again, issues of both guarantee and assurance are in evidence here, but it appears that the Green Book is, in the sections that we are about to examine, concerned largely with what we would term guarantee. Another section of the Green Book presents a graduated assurance measurement scheme that partly incorporates the rating of mechanisms, however, so there is some ambiguity as to how to interpret some of the discussion. In the following, we will ignore further guarantee/assurance distinctions, and present the issues just as the Green Book does.

For each of the eight security functions, the Green Book lists a series of topics relevant to the function which must be considered when forming a security policy. As well, a list of topics is given which may form a basis for evaluating relevant mechanisms. Below we discuss the security functions in turn.

2.3.1. Identification and Authentication

The identification and authentication requirement is discussed first since it is a prerequisite for the three functions following, concerning access control and audit.

Important issues in identification and authentication are:

- which subjects and objects require identification only,
- which subjects and objects require authentication as well,
- what circumstances necessitate identification and authentication, and
- actions to be taken on authentication failure.

The Green Book does not make, for many purposes, a clear distinction between subjects and objects. The term *object* is defined in a Glossary as an entity that "plays the passive role in the administration of rights", but the term *subject* is not defined. Let us define, for the time being, *subject* as "an active entity which seeks access to objects and other subjects." Subjects can be users or processes, there being no requirement that a process must act on a specific user's behalf.

It is a consequence of the Green Book's "Rights" discussion (see below) that subjects may seek access to other subjects. Clearly, in this case, the subjects being accessed must be processes. One could then attempt to split processes into subject and object components, and then make the statement "only subjects access only objects", but the Green Book does not address this detail.

It is interesting that the Green Book treats, according to the four issues listed above, identification and authentication as discrete, atomic operations, rather than viewing authentication as an integral, but variable-strength, component of identification. The strength of authentication is taken care of by the mechanism evaluation step discussed below. It appears that authentication is viewed as a binary design choice, with strength an assurance issue. However, we will discuss the mechanism rating here, since we feel that system designers want at least some flexibility in choosing authentication strength, and make other design choices based on it.

Also, the idea of authenticating objects, rather than merely identifying them, addresses the Labeling issue, and may be extended to include correctness and completeness checks on objects, addressing (at least a portion of) the Integrity issue. While authentication of a subject usually involves some exchange of information with the subject (possible due to the active nature of subjects), authentication of an object must be performed by inspecting the object's data. These sorts of considerations help to blur any distinctions one would like to make between identification and authentication.

Mechanisms implementing this function must be evaluated based on:

- the uniqueness offered by the identification scheme, and
- the possibility of authentication deception.

A guideline for rating an authentication mechanism based on uniqueness of the identification is provided. This is interesting because definite numerical values are given, which supports our hope that a quantifiable measure of guarantee is achievable.

In the following table, "P" represents a range of probabilities of incorrect identification, and "Rating" represents the best rating attainable over that range:

P	Rating
10^{-2} to 1	Weak
10^{-4} to 10^{-2}	Moderate
10^{-6} to 10^{-4}	Strong
10^{-8} to 10^{-6}	Very Strong

Thus, a "very strong" authentication mechanism must incorrectly identify fewer than one entity per million. For example, a user-authentication system which can distinguish among 1000 fingerprints is at best a "moderate" strength mechanism.

When evaluating the possibility of authentication deception, additional factors must be considered. These differ, depending on the authentication mechanism in use, which can take three forms. Authentication by ownership involves the presentation of some object (e.g. a hardware key) as proof of identity. Authentication by knowledge requires that some "fact" (e.g. a password), or "ability" (e.g. the computation of a secret algorithm) be presented as proof of identity. Authentication by characteristic features involves inspection of the subject or object for some unique, distinguishing characteristic (e.g. a fingerprint). These additional evaluation factors are listed below.

1. Authentication by Ownership

   - the possibility of theft of the object used for authentication
   - the effort required for forging the object

This mechanism is prone to theft or forgery, and the only protection against these threats is an additional authentication action. Also, the channel used for communication with the authentication object must be evaluated, and if the object contains intelligence (e.g. a chip card), it too must be rated as a component of the overall system.

2. Authentication by Knowledge

   - the effort required for obtaining the knowledge

Knowledge can be transferred intentionally or accidentally outside of the system, and it is possible to defeat this type of mechanism by trial and error. The number of guesses possible per unit time, and the necessary special knowledge must be determined to assess the trial-and-error threat. Again, an additional authentication action is the only protection, and the communication channels must be rated as well.

3. Authentication by Characteristic Features

   - the effort required for forging the characteristics

The features must be sufficiently unambiguous to differentiate among all possible identities, and firmly linked to the subject or object to be identified. Forgery is the main threat, and, if the characteristic features are not of an IT (information technology) nature, an external assessment of their suitability is required.

Clearly this discussion is oriented toward authentication of users, but many of these concepts can be applied to authentication of objects and processes as well. Authentication by ownership is best exemplified by the association of objects and processes with "capabilities" — large random bit-strings, essentially unforgeable, yet often prone to copying. If unauthorized passing of capabilities cannot be prevented, then authentication must be ensured by other means. The point is that authentication of users by ownership, or objects and processes by capabilities, is rarely foolproof unless very tight control can be maintained on the system's operating environment.

Authentication by knowledge is analogous to process authentication in the sense that password exchanges with users are analogous to challenge-response protocols in distributed systems. In fact, the greater computational power of processes relative to human users implies that authentication by "ability" may be better suited to processes. There is no natural analog for objects.

Authentication by characteristic features is essentially equivalent to authentication by ownership for objects and processes.

It is interesting that evaluation of this most fundamental of security requirements seems to require some form of composition of secure components. This is because authentication of subjects requires some sort of communication with peripherals for authenticating users or foreign processes. All elements on this communication path must be highly-trusted before the effectiveness of the authentication mechanism itself can be verified.

2.3.2. Administration of Rights

The Administration of Rights security function addresses the central issue of computer security: access control. The discussion in the Green Book is of a highly speculative nature, making it applicable to a very wide variety of access control requirements.

Subjects can possess rights with regard to other subjects and objects. The following may be relevant components of a system security policy:

- which subjects and objects (or classes of these) are subject to the administration of rights,
- which kinds of rights can exist between subjects and objects,
- who is entitled to grant or alter rights,
- what rules have to be enforced for the granting or alteration of rights,
- what conditions have to be fulfilled prior to the granting or alteration of rights,
- what roles have to be defined by the administration of rights,
- what rights are associated with special roles, and
- what roles are mutually incompatible.

The first component above corresponds to the Orange Book notion of trusted and untrusted portions of a single system. The TCB need not administer all subjects and objects, though one might expect that a highly-trusted system would have no untrusted portion, as required by Orange Book levels higher than C1.

The second component illustrates that the Administration of Rights security function encompasses the central concerns of computer security: access rights between subjects and objects. These "kinds of rights" include, for example:

- "reading" (non-modifying access) of objects by subjects,
- "writing" of objects by subjects,
- "appending" onto objects by subjects,
- "creation" of objects by subjects,
- "spawning" of new subjects by subjects, and
- "messaging" between subjects.

Clearly these rights have confidentiality and integrity concerns, but they also address a large portion of the availability problem, since part of guaranteeing availability is limiting access which can affect proper system functionality.

The third component, "who administers rights", is a crucial aspect of a security policy. In fact one of the prime distinctions between Discretionary and Mandatory Access Control mechanisms is the centralization of control enforced by MAC. In addition changes in the control structure produce a wide variety of DAC policies. Research has also lately considered the formulation of multi-person control policies, especially for the implementation of multiple-role mechanisms similar to those employed in certain commercial and military decision-making systems.

Rules for granting or altering rights encompass the access restrictions which are central to traditional security policies. Examples of access rules are the Simple Security and Star Properties of the MAC mechanism, the access-control list often used to implement DAC, and the validation requirements in the Clark-Wilson integrity mechanism.

The following component, on first glance, seems identical in intent. However, it is likely that the "conditions" in question are requirements on the operating environment at the time of rights administration. For example, changing the allowable rights on a particular set of objects may require that no subjects currently possess access rights to any of the objects. In some cases, altering the rights structure in certain ways may require that the system be brought into a "single-user" mode.

The final three components concern the function of "roles" in the administration of rights. The Green Book defines a role as "a grouping of rights allocated to a subject", which implies that, since rights exist between subjects and other subjects and objects, every subject has a "role" by definition. In fact this component is concerned with the possible requirement that certain "special" roles may need to be defined, for example, the roles of system administrator and system security officer. A policy may define these roles by specifying the rights the subjects in question have with respect to certain pre-defined subjects and objects, and to certain classes thereof. If a large number of roles is defined, some care must be taken that they are mutually compatible, in the sense that rights and responsibilities are partitioned in a consistent and reasonable manner.

Mechanisms implementing this function must be evaluated based on:

- the completeness of the administration of rights,
- the unambiguity and consistency of the structure of rights,
- the clarity of the structure of rights,
- the protection against covert changing of rights,
- the protection against development of unalterable rights relations,
- the administration of associated rights when deleting or renaming subjects or objects, and
- the protection against restrictions in the capability of exercising rights.

The "completeness" criterion is clearly the dual of the first policy component, concerning which subjects and objects are subject to rights administration. The intent appears to be that evaluation in this respect consists of verifying that the portion of the system which is designed to be trusted, is in fact administered by the implementation. Whether or not all accesses in this portion are properly restricted is a matter for the Verification of Rights function. Also, it must be demonstrated that all forms of rights can be granted as required by the security policy.

Unambiguity, consistency, and clarity are metrics of the practicality of a particular rights structure. Mandatory Access Controls based on levels of sensitivity are generally simple in structure, as long as the number of "need-to-know" categories is reasonably small. Discretionary Access Controls, on the other hand, can quickly lead to rights relations of great complexity and possibly unintentional structure, especially given a large number of users. If the structure of rights is such that the consequences of altering some right are not immediately obvious, the effectiveness of the mechanism is substantially weakened. In cases where complex rights structures must necessarily evolve (as would be the case in most real-world environments), some kind of automated support for the display and interpretation of rights relationships would be necessary.

The rights structure must itself be immune to covert alteration, of course, though, if the rights structure is viewed as an object, or set of objects, it could conceivably be protected by the normal mechanisms. If this were the case, there would remain the necessity for a higher-level protection scheme for certain rights data, since, in the limit, no protection mechanism can protect itself. However, it should be the case that this data is sufficiently simple and static in structure that very different, highly trusted, mechanisms would be available to safeguard it.

Unalterable rights relations may develop in exceptional circumstances, such as the loss of hardware keys, the simultaneous departure of critical security personnel, or machine malfunction. It must be possible in all cases to recover system functionality while maintaining secure operations. This will usually involve removing the system from normal use temporarily.

The deletion and renaming of subjects and objects has implications that must be dealt with, in similar fashion to the Orange Book "object reuse" requirements. Object Reuse is in fact a separate Green Book security function, to be discussed below.

Finally, the criterion dealing with restrictions in the exercising of rights is primarily an availability concern. It must be determined whether denying access (for example, due to the existence of multiple writing processes, or to avoid a deadlock situation), contravenes the security policy. The policy must be formulated with these possibilities in mind.

2.3.3. Verification of Rights

The Verification of Rights security function is concerned with the actual enforcement of access restrictions and other rights matters.

> The purpose of this security function is to permit subjects to exercise only the rights they have been granted. A security policy can specify:
>
> - which actions require a verification to be performed,
> - what should be done if an action is attempted without possessing the appropriate right, and
> - what exceptions shall exist to rights verification, and under what conditions.

On first glance, it would appear that all attempts at exercising rights should require a verification action. However, the inference here is that a security policy may specify which attempts need verifying. Two interpretations of this seem feasible.

First, attempts at exercising rights may be classified according to the level at which attempts are made. This is related to the particular set of objects and subjects which are considered subject to the administration of rights. For example, if "file" objects are protected, verification could be performed on file opens only, or on individual reads and writes. If smaller-grained objects, such as database records, were individually protected, a different set of considerations apply. If protection is enforced on memory segments, or protected variables, verification may conceivably be performed on nearly every machine instruction.

Second, rights verification may be implemented in two (or more) phases, as in the creation of an unforgeable ticket on a rights request, and the inspection of this ticket on subsequent attempts to exercise this right. Different forms (and different strengths) of verification are thus performed at each phase.

This discussion is a good example of the type of information presented in the Green Book. There is little in-depth consideration of any security issue, but there are many lists of topics, each of which require careful scrutiny if a large portion of the computer security spectrum is to be addressed.

The second policy component listed concerns the treatment subjects should receive if they attempt to exercise a right without its having been granted. Again, this can be interpreted in two ways.

First, if verification is implemented at a "low" level (for example, on every individual object access), it is not immediately obvious whether a failure of verification indicates a wilful attempt to bypass the rights mechanism. Repeated failure, however, of verification for closely related rights might indicate such an attempt.

Second, if rights verification is a two phase process, a subject may request some action, and then attempt to provide proof at some later time that this request was granted. A subject that provides an improper "proof" of right, must be considered to be either defective or malevolent. In both cases, it may be desirable to terminate the subject (assuming the subject is a process, of course; a user would have his or her session terminated).

Actions that are deemed to be attempts at willfully bypassing the administration of rights would likely be audited, at the least. This is discussed further under the Audit security function below.

The third policy component concerns exceptions to rights verification. It would appear that these exceptions must only be relevant under "exceptional" conditions, such as might exist upon machine failure, when perhaps any user accessing the console may do whatever is necessary to restore system function. These considerations appear to be addressed by the Error Recovery security function, discussed below. Other types of exceptions to rights verifications are addressed by the first policy component above.

Mechanisms implementing this function must be evaluated based on:

- the completeness of the verification of rights,
- the time at which the rights are verified,
- the availability of decision-making data, and
- the integrity of the decision-making data.

How verification of rights is accomplished depends to a large extent on the manner in which the policy has been formulated, but clearly ensuring that rights checks cannot be bypassed encompasses most of the work in contemporary trusted system evaluation. The verification mechanism is to be rated based on the effort required to bypass it. This is the completeness criterion - a perfect mechanism will effectively verify every attempt to exercise a right.

The second criterion concerns the timeliness of verification. Multiple-phase verification mechanisms, as discussed above, give rise to difficulties when changes are made to the rights structure. The revocation of an access right, for example, is complicated when subjects already possess proof of verification. This criterion is rated based on the length of time subjects may continue to exercise rights that have been withdrawn, and on the consequences this may have.

Availability of rights and authentication data is particularly a problem in distributed systems, and it must be ensured that availability policies are followed, without allowing the granting of rights given insufficient information. In general, satisfying this criterion involves choosing a reasonable compromise between performance and availability on the one hand, and trustworthiness on the other. A highly-trusted system would be expected to sacrifice performance and availability concerns in the interests of confidentiality, integrity, and authentication quality.

The final criterion listed concerns ensuring the integrity of data used to verify rights. This appears very similar to the criterion addressed under administration of rights, concerning protection against covert changing of rights. Here the emphasis is on protection of “lower-level” data than that representing the rights structure itself. This lower-level data includes authentication information (for subjects and objects) that is relevant to the verification of rights function, as well as temporary rights data, such as capabilities or cached access lists. Also considered here are integrity violations due to hardware and communication errors, and possibly also due to software or user (administrator) errors. The effect of such errors should be random alteration of the decision-making data, and the rating of the mechanism is thus based on the probability of error occurrence, and any correlation between probability of damage and importance of the data (for example, highly critical data should be practically immune to such alteration).

2.3.4. Audit

Auditing is the recording of security-relevant system events. The Orange Book audit requirements are quite stringent at all levels above C1, and are presented in great detail, whereas the discussion in the Green Book is brief and speculative.

> The audit function records information about attempted exercising of rights. This permits analysis of subject behaviour for the detection of successful or attempted misuse of rights. Policy formulation may include the following considerations:
>
> - which events shall be recorded,
> - what information shall be recorded,
> - where the audit information is to be stored,
> - criteria for obtaining access to audit information, and
> - how this information should be evaluated.

Certainly the first two considerations form the basis of audit quality evaluation. The fundamental requirement of auditing is to record sufficient information that dangerous situations can be recognized, and hopefully that the events leading to this situation can be unambiguously identified. At the same time, some consideration must be paid to the volume of data that is generated, and which must be stored reliably and economically. Satisfying these conflicting concerns requires a very careful study of the inter-relationships among system functions, to determine just which auditable (i.e. discrete and recognizable) actions are relevant to security functionality. This analysis is essentially the same as must be performed during the design process, in specifying the Administration and Verification of Rights policies and mechanisms. Thus, careful design or analysis should result in the determination of a sufficient amount of auditable information. What remains is to determine which information is necessary, to keep the audit volume manageable.

The next two considerations address the accessibility of audit information. Clearly, this data must be protected from unauthorized observation or modification, and likely can be stored on relatively low-performance media (e.g. magnetic tape). Also, the utility of audit information decreases rather quickly with time, and some scheme for disposing of, in some graduated manner, aging data may prove acceptable. Access to this data must, of course, be subject to the usual "need-to-know" restrictions, and may in fact be confined to certain trusted programs. The Confidentiality and Integrity mechanisms available should be adequate to protect audit information.

The evaluation of audit information can be specified in a wide variety of forms. Along with the practical matter of compressing and disposing of aging data, it is really the only aspect of the Audit security function which is not subsumed by proper consideration of the other security functions. Manual perusal by trained personnel is the simplest solution, but is clearly inadequate given the volume of information likely to be generated even by a modest-sized system.

Effective use of audit data requires continuous monitoring by processes endeavouring to detect patterns of resource usage or attempted rights exercise indicating a potential rights violation. The design of such programs is likely to involve specialized skills quite distinct from those used to define other aspects of security policy and mechanism. However, there may be considerable overlap between the algorithms used to prove formal security properties, and those used to analyze audit trails. Real-time audit processing would allow much more flexible access controls to be implemented, since individual rights decisions would not have to be based on worst-case scenarios - rather, the exercise of a particular right could be monitored as it evolves. It is unlikely, however, that this ideal will be realizable in the near future.

Mechanisms implementing this function must be evaluated based on:

- the non-deceivability of the auditing, and
- the completeness of auditing.

These criteria address, respectively, the possibilities of recording events that did not take place, and of failing to record events that should be audited. The former is clearly a form of Integrity concern, whereas the latter is a by-now familiar completeness criterion. The mechanism must guarantee that information is not lost due to media capacity limitations, and must ensure that all information crucial to the proper interpretation of the audit data is recorded, such as the spawning and termination of audit-interpretation processes.

2.3.5. Object Reuse

Reusable objects are often dealt with specially since they are subject to very different performance constraints than are named objects, handled by the Verification of Rights function. Reusable objects may include main memory segments, input-output buffers, and machine registers.

Reusable system resources must be treated in such a way that no flow of information occurs which violates the security policy. Relevant aspects of a security policy include:

- what resources shall be treated, and
- which actions necessitate special treatment before or after use.

The Green Book treats only the Confidentiality aspects of object reuse, though there are Availability concerns here as well. In fact, meeting Availability constraints is largely accomplished by controlling the sharing of reusable system resources.

In this light, it is clear that the resources requiring treatment prior to reuse are those which may contain data sensitive to disclosure, and which are not dealt with satisfactorily by the Verification of Rights mechanisms. The precise definition of this set of resources must occur during successive policy-mechanism mappings.

One aspect not mentioned here concerns the nature of the treatment required for reusable objects; that is, how these objects should be cleansed of sensitive information. This is addressed by the mechanism evaluation criteria below, but its appearance there, rather than as a policy component, illustrates the apparent ambiguity between policy and mechanism, since the treatment might well be specified by the policy. This ambiguity will be well explored later, when we rigourously define our notions of guarantee and assurance.

Mechanisms implementing this function must be evaluated based on:

- the nature of treatment for reusable objects, and
- the timeliness of treatment.

The effectiveness of the treatment applied to reusable objects must be determined, by measuring, in some way, the volume and sensitivity of information that may remain for possible unauthorized disclosure. As well, a completeness check must be made, to ensure that all objects requiring treatment do in fact receive it. The mechanism rating is based on the effort required for a subject to access unauthorized information, either due to inadequate cleansing, or to failure to treat some reusable object.

The timeliness criterion is a related concern, arising due to the possible presence of alternative paths by which to access objects prior to treatment for reuse. Satisfying this criterion is thus a matter of demonstrating that no such alternative paths exist, at least until the object has been released to the reusing subject.

The aim of the Object Reuse security function is to preserve Confidentiality given the finite physical resources of an implementation. As such it is really an unavoidable consequence of performing the low-level policy-mechanism mappings when proceeding from design to implementation. One would thus expect it to be subsumed by the Verification of Rights function. The potential for abuse in the form of covert channel establishment necessitates special-purpose mechanisms, however.

Similar observations hold for the following three security functions. They are specifications for the enforcement of rights relations given the limitations of physical computer systems. This is mentioned in the introduction:

> The first four functions are derived from considerations of the term “right”. The remaining are necessary since systems are run on technical equipment which firstly cannot be error-free and secondly possesses finite resources.

It is interesting that the Green Book groups security concerns into these particular functions, without giving any justification for doing so. Later we will re-examine this question and provide our own classification of security concerns, in a form allowing meaningful measurement of guarantee and assurance.

2.3.6. Error Recovery

The Error Recovery security function addresses concerns which can be considered to be “fault tolerance” issues, but restricts its scope to those aspects of fault tolerance that are directly security-related. This function is intended to ensure that rights mechanisms cannot be bypassed in the event of a system malfunction.

> The error recovery function should minimize the impact of errors on the system, ensuring continuous secure operation. The basic requirement is for errors to be detected and dealt with as early as possible. The security policy can specify:
>
> - what errors shall be recovered,
> - what mode of operation does the error recovery result in (e.g. controlled termination, checkpoint and restart), and
> - what impairment (e.g. loss of data, functionality, or time) can be tolerated.

As for the Object Reuse function, the specification of detailed requirements for Error Recovery is problematic, since the system behaviour in this respect is highly implementation-dependent. Thus, only broad classes of recoverable errors would likely be enumerated in a security policy.

The occurrence of an error is likely to render the system less reliable in enforcing rights restrictions, and this must be compensated for by reducing the offered functionality. We might therefore expect to define classes of system functionality corresponding to the classes of recoverable errors which have occurred. These classes may be further refined as a system design progresses from high-level policy to low-level mechanism.

The third policy component addresses tolerable system impairment, which would seem to limit the scope of recoverable errors that can be tolerated within each functionality class.

Mechanisms implementing this function must be evaluated based on:

- the completeness of the error identification,
- the correctness of the error analysis,
- possible loss of data, functionality or timeliness,
- independence of the recovery procedure from the error source, and
- possible sources of error in the actual error recovery.

Error recovery must be preceded by an error identification step, aimed at ascertaining the nature and extent of the system impairment. The completeness criterion measures the coverage of the error identification mechanism, and one would expect that this coverage can be very high only when broad classes of errors are considered. The correctness of error identification is especially important when some sort of recovery is to be attempted; that is, when it has been determined that the fault is to some extent recoverable. Then, sufficient information must be gathered to allow a reliable recovery. The rating for these mechanisms must, according to the Green Book, be based on the probability of weaknesses in their implementations.

A quite different basis for rating error recovery mechanisms is examining the quality of the functionality classes entered after an error has been encountered. This examination is restricted, again, to the security-relevant aspects of system functionality, especially Integrity and Availability. The rating is derived by considering the possible decrease in security functionality, weighted by the probability of an error leading to this class. Computing this quantity is thus similar to comparing the security functionalities of two different systems, the feasibility of which we are trying to establish by formalizing guarantee and assurance.

The final two criteria measure the quality of the error recovery process. Clearly, the portion of the system devoted to error recovery must be highly-resistant to errors occurring inside or outside the error recovery module. Establishing ratings for these criteria is likely to involve substantial measurement of hardware and software quality.

2.3.7. Continuity of Service

The Continuity of Service function is closely related to the Error Recovery function, but deals with those portions of the system that ensure correct operation in the absence of errors. These include deadlock avoidance algorithms, real-time response guarantees, resource sharing policies, and other functionalities that are largely concerned with meeting Availability requirements.

Certain critical system functionalities must be maintained at all times for an acceptable level of security. A security policy can specify:

- what functionality shall be maintained with which priority,
- boundary conditions within which certain functionalities shall be maintained,
- circumstances under which the loss of certain functionalities can be tolerated.

These policy components together compose a “liveness” property for Availability. The loss of any system functionality impacts Availability and may thus be a transgression of the security policy, even though other security requirements, for example Confidentiality, remain satisfied.

It would appear that these requirements are in fact structured rights that should be handled by the Administration and Verification of Rights functions, but, as was the case for Object Reuse, the Green Book authors chose to create a separate security function to address these issues. One might attribute this to the fact that many of these functionalities affect the rights of subjects only indirectly. Thus, mechanism requirements derived solely from the consideration of rights structures may be insufficient to enforce these practical functionality requirements. However, since we are concerned here with security functionality only, and the Availability requirements may be formulated as structured rights relations, the Continuity of Service policy components do not appear to be necessary.

As we will see later, the Harmonized European Criteria address this issue by separating the Availability requirement from Confidentiality and Integrity, and integrating it with the Continuity of Service function, suggesting that access control mechanisms are not well suited to ensuring Availability.

Mechanisms implementing this function must be evaluated based on:

- the sufficiency of mechanisms for identifying error-prone situations, and
- the side-effects of the error avoidance mechanisms.

Similarly to the completeness criterion for Error Recovery, mechanisms for Continuity of Service must be rated based on the coverage achieved when identifying situations which may compromise some aspect of system functionality. The rating should take into account the probabilities of occurrence for each class of service-disruptive situation. Also, the effects on security functionality of any error avoidance measures must be assessed.

Thus, it may be advisable to establish certain classes of threats to service continuity, and corresponding levels of security functionality which can be provided while these threats are being actively avoided. Rating mechanisms on this basis involves, as for the Error Recovery function, a comparison of guarantee and assurance measures at each level. A trusted system may ultimately be labeled, then, by a set of guarantee-assurance levels, each of which it can provide at a corresponding level of non-security functionality.

2.3.8. Data Communication Security

It is generally not possible to protect communication channels completely via the aforementioned mechanisms for Administration and Verification of Rights. The Green Book provides a sub-classification of security functions that address the distinctive problems of secure data transfer.

The Orange Book does not place the same emphasis on data communication security, although similar considerations are dealt with in the Trusted Network Interpretation, which applies the Orange Book's treatment of single-computer security issues in a network setting. The important difference is that the Green Book recognizes that data communication security is of great importance even when the system is a single machine. This is due to the requirement for connection with input/output peripherals whose operation cannot be closely monitored by the central machine's TCB.

The Orange Book contains specifications for the behaviour of input-output channels carrying sensitive information, but does not deal with the subtleties of secure information transfer in a comprehensive manner. This, in our opinion, is one of the major strengths of the Green Book's approach.

The following sub-functions, while overlapping somewhat with the basic security functions, address the particular requirements for secure data communications. We will discuss each very briefly. This classification scheme is based on the Security Addendum to the ISO Open Systems Interconnection Reference Model (ISO 7498-2-1988).

2.3.8.1. Peer Entity Authentication

The purpose of this function is to assure communicating parties of the validity of each other's identification. It is not clear how this differs from the Data Origin Authentication function below.

The individual security policy and mechanism components listed under "Identification and Authentication" apply here.

2.3.8.2. Access Control

This function concerns control of access to system resources which may be used for data communications, in order to prevent the establishment of unauthorized information paths. Protection of the communicated data itself is the province of the following two functions.

The individual security policy and mechanism components listed under "Administration of Rights" and "Verification of Rights" apply here.

2.3.8.3. Data Confidentiality

This function addresses the requirements for confidentiality of communicated data.

The security policy can specify:

- for which channels confidentiality mechanisms are necessary, and
- which data must be protected in which channels.

Two factors complicate the application of "normal" policies and mechanisms to the control of access to communicated data. First, it is often appropriate to employ quite different mechanisms from those used to protect data maintained within a single system (under the control of a central TCB). Second, practical communication involves the generation and transfer of additional data that must be protected, such as address and protocol control fields, and communication patterns.

Mechanisms shall be evaluated under the following aspects:

- protection of the data transmitted by the various protection mechanisms along the entire communication path,
- protection of additional transmitted data that is itself intended to safeguard the confidentiality and integrity of the data, and
- proof that an authorized recipient can interpret the data.

Assuming encryption is used as a confidentiality mechanism, then the rating is to be based on encryption strength, effectiveness of cryptographic checksums, and reliability of key distribution protocols, roughly corresponding to the three aspects listed.

2.3.8.4. Data Integrity

This function addresses the requirements for integrity of communicated data, which may differ substantially from requirements on data maintained within a single machine.

The individual security policy and mechanism components listed under "Error Recovery" and "Continuity of Service" apply here. In particular, mechanisms shall be evaluated on the basis of:

- the integrity of individual data packets or data fields,
- the integrity of the entire data stream, and
- the integrity of the sequence of individual packets.

Interestingly, the Green Book claims that the integrity of communicated data can be assured by the same means employed for the Error Recovery and Continuity of Service functions, claiming that integrity can be achieved by error identification, recovery, and avoidance mechanisms. Although there may be considerable overlap among the requirements for Integrity, Error Recovery, and Continuity of Service, it appears questionable at this point to claim that they should be addressed by the same set of security policy components, since there is surely some aspect of each function which is irrelevant to the others (at least for the purpose of formulating a security policy.

2.3.8.5. Data Origin Authentication

Authenticating the source of incoming data is clearly a concern that is largely addressed by the Identification and Authentication function.

The individual security policy and mechanism components listed under "Identification and Authentication" apply here.

In fact, one could reasonably claim that this function is completely subsumed by the Identification and Authentication function, since any system must have communication facilities by which users gain access, and thereby requirements for user authentication. The existence of networks of systems does complicate this matter, however, since one system may be required to trust that authentication on another system is sufficiently strong. Also, the authentication of foreign processes may differ substantially, in policy and mechanism, from the authentication of human users.

2.3.8.6. Non-repudiation

This function deals with the requirement for proof that some data exchange has occurred. First, the recipient of data may need to verify the identity of the claimed sender. Second, the sender of data may need to verify that the intended recipient did indeed receive the data.

The individual security policy components listed under "Identification and Authentication" and "Audit" apply here.

Audit requirements are mentioned here because, when recording the transmission or receipt of data, some proof must be obtained that the transfer occurred as claimed. Without such proof, auditing in a distributed environment has little value.

Mechanisms implementing this function must be evaluated based on:

- verifiability that a particular data stream was transmitted by a particular subject, and
- proof that a particular data stream was accepted by the proper recipient.

The principal concerns of this function are authentication-related, and mainly require that communicated data and the sender and receiver fields be separable only by the sender and receiver. This prevents eavesdroppers from learning traffic patterns and authentication information by examining message contents, then using this information to masquerade as some other subject. The requirement for proof of proper message delivery effectively constrains protocols to require a notice of acceptance from recipient to sender.

2.3.9. Conclusions

The scope of the Green Book's consideration of guarantee issues is quite vast, but each issue is explored in little depth. As examples of more detailed investigation of the issues, ten Functionality Classes are defined, each defining a system at a different level of guarantee. In fact the intent of the Green Book is not to define a particular set of security functionality classes against which real world systems (in the design or implementation phase) are to be compared. The intent is that, by organizing the intended and actual security-relevant capabilities in the framework provided by the eight security functions, a guarantee measure can be defined for any system, which can then be used to judge its suitability for a particular application.

The first five classes are intended to exhibit a progression of increasing guarantee, each class a proper superset of its predecessor in terms of security functionality. These five classes correspond to the Orange Book classes C1 through B2, and thus serve as examples of how the Green Book's guarantee concepts map onto the Orange Book's.

The remaining five Functionality Classes present more specialized requirements for less general-purpose systems. For example, one class lists policy components only for the Error Recovery and Continuity of Service security functions, and claims to be suitable for systems with stringent Availability requirements. This class, however, illustrates some of the problems of the "decoupled" approach to guarantee specification. On the one hand, if this system allows users to access it, then there must be some requirements regarding Administration and Verification of Rights, as well as Identification and Authentication. On the other hand, if no users are allowed access, the system can hardly need a security policy, and the Error Recovery and Continuity of Service requirements are fault tolerance concerns, not security concerns.

2.4. THE HARMONIZED SECURITY CRITERIA

In May 1990 four European nations collaborated to produce a set of security criteria that consolidated and harmonized their own national computer security criteria. The draft form of these criteria, prepared by representatives of the governments of France, Germany, the Netherlands, and the United Kingdom, has been widely distributed for comment. We are interested here in the contribution this document can make to our understanding of the concept of guarantee.

Although the Harmonized Criteria includes material from the German Green Book, there is in fact relatively little discussion of guarantee. Whereas the Green Book put nearly equal emphasis on guarantee and assurance criteria, the Harmonized Criteria leans heavily toward the assurance aspect. This may be due to the belief that requirements on security functionality (guarantee) are highly variable, depending on the intended system application, the operating environment, the current state of the art in software and hardware, and so on. Moreover, there is implicitly expressed the notion that the provision of assurance in a system's proper operation is a task largely independent of the nature of the security functionality in question. Thus, the Harmonized Criteria mainly present a methodology for evaluating security functionality, essentially regardless of what that functionality is.

In fact, almost none of the assurance criteria address issues restricted to computer security. That is, the large majority of assurance measures described can apply to assurance of any functionality. This is not a weakness of the Harmonized Criteria, but it is remarkable that approximately seventy percent of the material described is not security specific.

Security functionality is treated in the Harmonized Criteria at three levels of abstraction:

- Security Objectives - Why the functionality is wanted.
- Security Functions - What is actually done.
- Security Mechanisms - How it is done.

The security objectives are expected to be drawn from the basic requirements for Confidentiality, Integrity, and Availability, and form some high-level policy, independent of system function. The security functions are more detailed requirements on the particular system in question, and are to be stated by a system's developer as a set of claims. The security mechanisms are specific system components, whose capabilities for enforcing the security functions must be demonstrated by a system evaluator.

As in our investigation of the Green Book, our primary interest here is in the organization and content of the Security Functions, and the measures proposed to evaluate the Security Mechanisms. The Harmonized Criteria recommends eight Security Functions, as does the Green Book, but organized somewhat differently. The treatment of each function is quite brief, and little mention is made of specific mechanisms, so our discussion of the Harmonized Criteria will also be brief. We deal with each function in turn below, emphasizing differences from the corresponding treatment in the Green Book.

2.4.1. Identification and Authentication

This heading should cover any functions intended to establish and verify the claimed identity of a user. These include:

- authentication of users,
- addition and deletion of authorized users, and
- modification of authentication information.

In contrast with the Green Book, no mention is made here of authenticating processes or objects; presumably, this is subsumed by the following two functions, Access Control and Accountability, as will be discussed.

2.4.2. Access Control

This heading should cover any functions intended to control the flow of information between, and the use of resources by, users, processes, and objects. These include:

- administration of access based on authorization and need-to-know,
- maintenance of lists or rule bases governing access rights, and
- temporary restriction of access to maintain the consistency and accuracy of shared objects.

This function corresponds to the Green Book Administration and Verification of Rights functions. This seems to be a reasonable consolidation, since the two functions are necessarily closely related, in that alteration in the policies or mechanisms of one would likely give rise to alteration in the other.

As noted above, identification and authentication of objects may be seen to be addressed by the second point, while the third point addresses aspects of Availability, Integrity, and fault tolerance.

Access control is the central issue in computer security, and thus countless issues could have been addressed. The treatment here is simplistic in the extreme, illustrating that assurance issues, largely independent of security considerations, are the main thrust of the Harmonized Criteria.

2.4.3. Accountability

This heading should cover any functions intended to record the exercising of rights to perform security-relevant actions. These include:

- collection of information such that users can later be held accountable for their actions,
- protection of this information, and
- analysis of this information.

There is considerable overlap between this function and the following one, Audit, which is somewhat puzzling. One might assume that the information discussed here is meant to include such items as process identities and capabilities, which record the identity (ultimately) of users in order that rights can be administered and verified at the time of their attempted exercise. Thus, authentication of processes is addressed, as are mechanisms for identifying users, and protecting this information from forgery or improper use. However, the thrust of this function appears to be the recording of information for later analysis, which is essentially the purpose of the Audit function.

2.4.4. Audit

This heading should cover any functions intended to detect and investigate events that might represent a threat to security. These include:

- collection of information such that later investigation can determine if security violations have occurred,
- determination of resources for which access restrictions have been compromised,

- protection of the collected information, and
- analysis of this information.

Again, it is not clear why this function has been distinguished from the Accountability function.

2.4.5. Object Reuse

This heading should cover any functions intended to control the reuse of data objects for the purpose of preventing unauthorized flows of information. These include:

- initialization of unallocated data objects, and
- clearing reallocated data objects.

The Object Reuse function deals with the application of access restrictions to an implementation-dependent system component, and thus seems somewhat redundant given the extremely abstract notion of access control employed in the Harmonized Criteria. However, mechanisms employed for this function may have very different performance requirements than more general mechanisms. The likely reason for the inclusion of this security function is the potential for the establishment of covert storage and timing channels through coordinated use of shared data objects.

2.4.6. Accuracy

This heading should cover any functions intended to ensure the correctness and consistency of security-relevant information. These include:

- determination of the relationships between related pieces of information,
- establishing and maintaining these relationships,
- detection or prevention of loss, addition, or alteration of information transferred between processes, users, and objects, and
- detection or prevention of alteration of claimed or actual source and destination for the transfer.

This function addresses two aspects of the Integrity objective. A third aspect, that of controlling write access to information, is addressed by the Access Control function. The Accuracy function is concerned with data correctness.

The first aspect is often referred to as maintaining "internal consistency", which involves preserving certain relationships among interdependent data. This is a complex and imperfectly understood problem, of which no more is stated. The notion of "external consistency", which is concerned with relationships among information both inside and outside the domain of the computer system, is not addressed here.

The second aspect is essentially a correctness requirement on the operating system, that it can manipulate data being transferred without introducing errors, and that internal objects used to hold this data are adequately protected.

2.4.7. Reliability of Service

This heading should cover any functions intended to ensure consistency of service and availability of service. These include:

- performance of time-critical tasks at their required time,
- prevention of non-time-critical tasks from becoming time-critical,
- ensuring that access to resources is possible when needed,
- prevention of unnecessary acquisition or retention of resources,
- detection of errors that may disrupt service, and
- recovery from such errors.

This function consolidates the Green Book's Error Recovery and Continuity of Service security functions, and addresses most of the concerns associated with the Availability objective. The central concern is to provide "fair" access to resources when allowable. That is, access rights should be granted when desired, unless this access unduly infringes others' rights. The two requirements addressing time-critical tasks are simply applications of this concern to the computational resources of the system. That is, the CPU and related mechanisms must be considered resources to be shared in a fair manner.

Besides the issue of fair resource sharing, there is addressed the problem of operational errors, whose occurrence may lower the availability of the system for all users. As noted in the Green Book, this problem can be split into error detection and error recovery phases.

2.4.8. Data Exchange

This heading should cover any functions intended to ensure the security of data during transmission over communication channels. These functions should be categorized under this heading, even when they could have been categorized under previous headings, since different types of mechanisms are usually employed in this case. These functions should be described using the terminology of the ISO Open Systems Interconnection Security Architecture (ISO 7498-2-1988).

This is a clear reference to the Data Communication Security function in the Green Book, which is also organized around the OSI terminology. No discussion of particular functionality is given, but it is interesting that communication-related security concerns are considered distinct enough that a separate function is reserved for them.

2.5. CONCLUSIONS

The Harmonized Criteria does not provide an in-depth discussion of security functionality, but emphasizes assurance issues, which are presented in a form largely independent of the particular security functionality supported by the system. As far as guarantee is concerned, the important contribution of the Harmonized Criteria is the organization of general guarantee categories, which differs slightly from that in the Green Book, and which may be considered to be a refinement thereof.

3 GUARANTEE MEASUREMENT AND COMPOSITION

Fear not to touch the best,
The truth shall be thy warrant.
Sir Walter Ralegh,
The Lie,
1595.

Since the measuring device has been constructed by the observer ...
we have to remember that what we observe is not nature in itself
but nature exposed to our method of questioning.
Werner Heisenberg,
Physics and Philosophy,
1958.

3.1. INTRODUCTION

Formal modelling techniques offer the benefits of precision of expression of the problem and its solution, as well as precision of elaboration of the chain of reasoning leading from suppositions to conclusions. However, there is a price to be paid for this precision, because of the necessity for casting the problem in a form amenable to analysis and compatible with available theoretical tools. This invariably seems to involve a certain amount of ‘‘shoehorning’’, occasionally taken to the extreme of discarding unmanageable ‘‘toes’’, with the result that the precisely expressed and precisely solved problem may not resemble the original problem as closely as we would like.

In contrast, it is possible to approach many problems in a less formal but more flexible way, relying on intuition and informal argument. The expression may not be as crisp, and the rationale not as objective, but by this method it is possible to deal with the practical aspects of a problem that the more formal methods cannot adequately comprehend.

We will develop such an approach to the measurement and composition of guarantee and assurance. We take as a fundamental principle of this work that the security issues that affect the trustworthiness of subsystems and of composite systems may be profitably divided into two groups, those relating to *guarantee*, which refers to properties of the system specification, and and those relating to *assurance*, which concerns our level of confidence that the implemented system conforms to the guaranteed specification. We will begin by examining and refining these definitions to yield the most advantageous division of issues.

The methodology developed here will be influenced by the notions of guarantee contained in the published security criteria documents of several countries. This effort thereby serves the dual purposes of providing an alternate approach to the composition problems as well as providing guidance, in the form of a target based on official policy statements, for our formal modelling work.

3.2. GUARANTEE AND ASSURANCE

The preceding chapter examined the question of categorizing security concerns as being related either to guarantee or to assurance. In distinguishing the two categories we have tried to apply a consistent set of definitions. The statement of work issued for this research contained definitions of these terms, and required investigation of the concepts embodied in the definitions. A guarantee was defined as

> an assertion that a given security functionality is sufficient to satisfy the requirements of a specified security policy

while assurance was

> the level of confidence that a guarantee associated with a specified security policy is both valid and effective and that assurance is associated with both the quality and durability of the guarantee

In these definitions, the term *security policy* is considered to designate

> the set of laws, rules, and practices defining the protection of a system and the information it contains from unauthorized disclosure, modification, destruction, or removal and protection from the interruption of services that the system provides

Our objectives are to develop methods of measuring guarantee and assurance, and methods for deriving the guarantee and assurance of a composed system from the guarantees and assurances of its components.

3.2.1. Applicability of Terminology

In the security area it is conventional to make fairly narrow interpretations of such terms as ‘‘functionality’’ and ‘‘policy’’ that appear in the definitions above. Without detracting from the utility of these standard interpretations, we suggest that there is also value in taking a more liberal view of these concepts and the relationships between them.

The process of designing and building a system can be viewed as a succession of refinements, beginning with a statement of overall objectives and ending with an instance of a mechanism constructed to achieve those objectives. The number of steps involved may vary, depending on the nature of the problem, the size of the project, or on the number of distinct groups involved in the process at different levels. It may even depend on who is doing the counting, as different observers may see varying numbers of informal refinement steps occurring in the same development process.

Still, it is possible to identify gross levels of refinement that should be acceptable to most readers. The process begins with a recognition of high-level **objectives**, usually as a collection of somewhat independent goals dealing with such issues as performance, cost, reliability, and control over the dissemination of information.

Consideration of these goals provokes and influences the establishment of **requirements** or **policies**, which are no more nor less than high-level design decisions expressing a large-scale strategy for achieving the specified objectives. The policy makers have some expectation, whether derived from

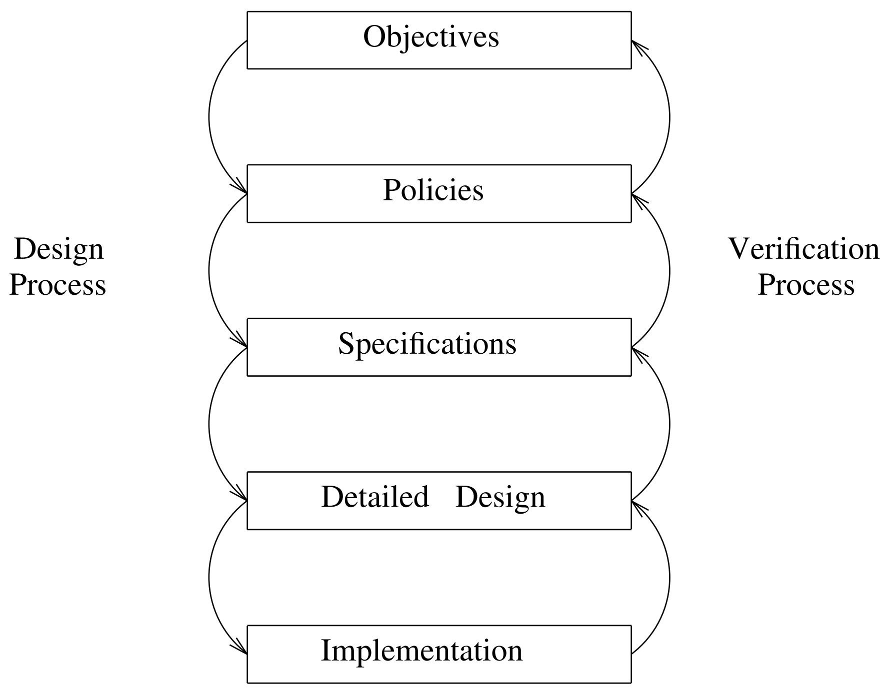


Figure 3.1: Refinements in System Design.

analysis or experience, that the enforcement of the policies will cause the goals to be met.

In the next step, often referred to as ‘‘high-level design’’ or, more increasingly, by such non-words as ‘‘architecting’’, a particular system is specified in sufficient detail to describe what it does, but not how it does it. This step is undertaken subject to the guidance provided by the policies and requirements established earlier. There is often a certain amount of compromise necessary here, since the several statements of policy and requirements may result in conflicting constraints on the specification.

The next refinement is the one conventionally known as ‘‘design’’, where an actual implementation is planned in detail. The specifications from the previous step define the target for this design, and success and failure of the design are largely defined in terms of meeting or failing to meet them.

Finally, the last step is the manufacturing, installation, operation, and upkeep of the system as a faithful rendering of the detailed design. Quality control, both over the raw materials and their processing, as well as the adequacy of the supplied documentation and of the prescribed maintenance procedures, are of prime importance here.

These five levels of specification are, in many cases, created by different organizations, or by different groups within an organization. They also deal with very different issues, and use different tools and notations in their production and presentation. Yet, the refinement steps and the relationships between them can be seen to share these qualities:

1) The refinements progress from the abstract to the concrete. Concepts at the higher level are more likely to be qualitative and subjective, while the lower level deals more with quantities and with specific objects.

2) Generalities at the higher level are made more specific at the lower. This may happen because a more narrowly defined range of problems is being addressed at the lower level which causes new application-specific requirements to be added, or because a more detailed view of the system makes visible low-level alternatives that were hidden at the higher level, or simply because the desired level of definiteness in the refined design requires that issues must be decided that were previously left open.

3) For each adjacent pair of levels there is some belief that the goal represented by the higher-level statement will be adequately met by its lower-level refinement. This belief may derive from experience, intuition, or confidence in the methodology employed, and may be augmented by the use of design verification techniques where such exist.

The cumulative effect of these refinements is to create a continuous derivation path from the statement of objectives to the concrete implementation, and to create a complementary ‘‘chain of belief’’ in the other direction. Although the language of the previously quoted definitions of guarantee and assurance seem to limit their application to the step between policy and specification, we believe it is both appropriate and useful to recognize that the successful outcome of this process requires that adequate guarantees and assurances be established from the implementation level all the way up to the original statement of objectives.

One way to try to accomplish this goal is to extend the range of the verification methods. For example, in a software verification system one can represent both a software implementation of a module and a specification of that module. Theorem proving techniques are then used to prove that the module implements its specification. Such a system might be extended to also permit the description of a Bell-LaPadula style security model, which could make it possible to demonstrate that the specification conforms to a particular security policy.

Another method that has attracted some amount of attention in the security community in recent years is the development of what might be called security ‘‘meta-models’’, which model security policies and objectives and are used to demonstrate the relationships that hold between them. An early example of this is the model of Goguen and Meseguer [Goguen82] in which the objective is stated as a non-interference condition from high to low users, while the policy is represented as a set of conditions on states and transitions of a state machine. An unwinding theorem serves to connect the two and provide the desired assurance that enforcing the policy will achieve the stated objective.

Succeeding sections of this report will concentrate primarily on the versions of guarantee and assurance previously defined, those being the ones that relate specification to policy. However, the reader should keep in mind that the strength of the chain of belief depends on the quality of all of its links, and be aware that statements made concerning the relationship of specification to policy, and the guarantee and assurance that this involves, will also have validity in the more general setting of arbitrary pairs of levels.

3.2.2. Multiple Refinement Design

Acceptance of the Objectives—Policies—Specifications—Design—Implementation paradigm also sheds some light on a difficulty that is often encountered when attempting to distinguish between guarantee (or functionality) and assurance.

In some cases, the distinction is an easy one to make, and the separation of issues it supports is clearly beneficial. There is, for example, a clear difference in functionality between a system that offers mandatory access controls and one that does not. Also, the level of assurance provided by a group of implementors who are handed a specification and simply expected to follow it is clearly not equal to one in which a separate verification team formally or semi-formally establishes a correspondence between the specification and critical components of the implementation.

It is when trying to progress past these easily distinguished cases that we encounter aspects of system security that are hard to classify because they display characteristics of both guarantee and assurance, or where our definition of assurance fails to apply to factors that are commonly held to be assurance-related.

The conventional intuitive understanding of assurance is one that relates primarily to the methods used in the design and construction of a system, rather than the characteristics of the system so designed. This may not be the situation with our proposed definition of assurance, because it intimately ties an assurance to a corresponding guarantee.

That there should be some type of interaction is probably unavoidable, since providing an assurance is meaningless unless we also describe what is being assured. Still, we would like to develop a notion of assurance (and a measure of it) that operates independently of guarantee. It should allow us to make a meaningful comparison between the assurances of two different guarantees, based on the levels of effort, sophistication, and control exercised in the design and verification. In particular, if substantially the same techniques, procedures, and controls are employed in both cases, we should conclude that the assurance levels of the two guarantees are essentially identical, whatever the details of the policies and implementations.

Consider, for example, the problem of user identification and authentication. Passwords, ‘‘smart card’’ keys, and biometric devices represent a range of cost, ease of use, and effectiveness. A specification of the system will indicate an authentication mechanism and how it is to operate, and so the choice seems at first glance to be one of differing guarantees, and, in fact, probably different levels of a family of related guarantees.

Looked at from another viewpoint, however, the choice of which technique to use can be considered to be an assurance question, since it affects the reliability of an identification guarantee. This is a troubling observation, because one of the prime benefits that usually flows from separating a problem into its component parts is that understanding and analyzing the problem becomes much simpler. If we are unable to cleanly separate guarantee from assurance, these benefits will not be realized.

Figure 3.2 presents this situation diagrammatically, using the same five-layer breakdown of the design process as before. We assume in this analysis that the type of procedure acceptable for user authentication is considered a policy issue. There is a common element to all three approaches, which is represented in the diagram by a single system objective, that the system accurately identify users.

Let us assume that three systems are designed and built to the requirements of the three policies, and that ‘‘substantially the same techniques, procedures, and controls are employed’’ in the design and verification of the Policy—Specification, Specification—Detailed Design, and Detailed Design—Implementation correspondences for all three systems. Let us further

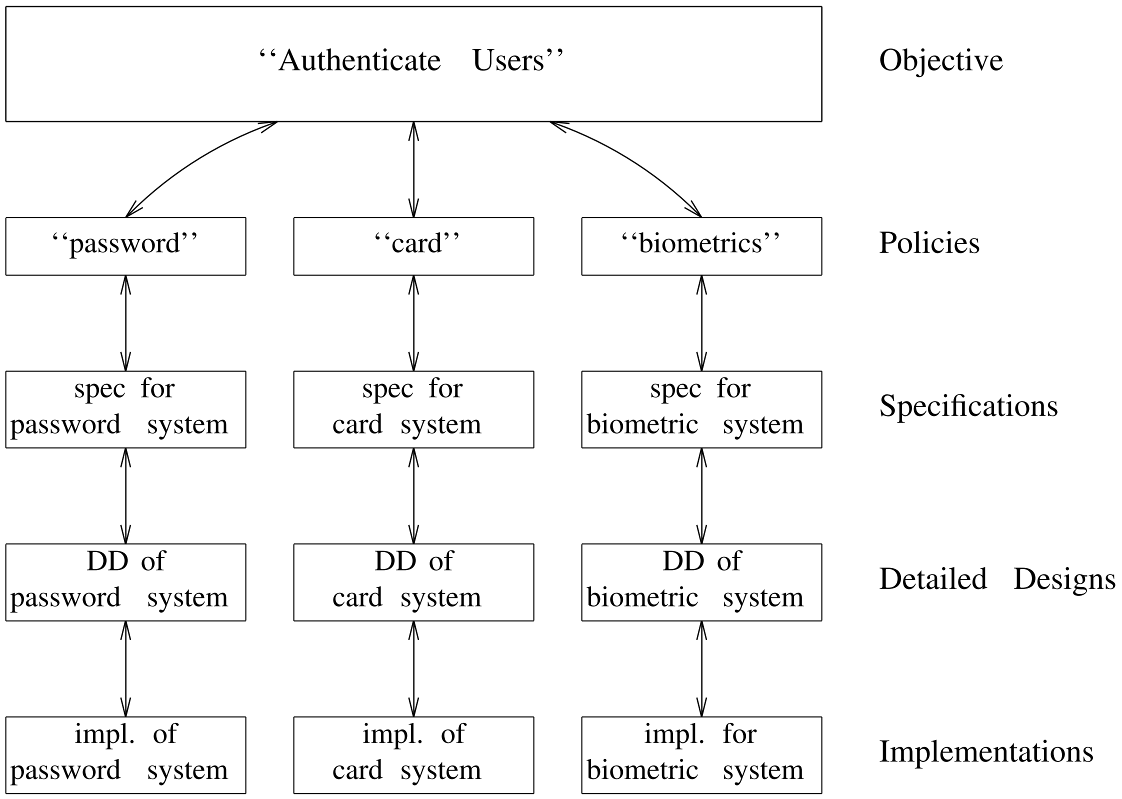


Figure 3.2: Differing Levels of Policy for a Single Objective.

assume that we have succeeded in developing a metric for assurance that behaves in an acceptable manner and which, in this instance, tells us that the assurances of all three Policy—Specification guarantees are equal, and similarly that the assurances of all three Specification—Detailed Design and Detailed Design—Implementation guarantees are equal.

Now, consider the overall guarantees of the three systems, that is, the composite guarantees from the Implementation to the Objective. Each is formed from a chain of guarantees leading to the same objective, and so are all of the form

Implementation *X* **satisfies the** *Authenticate Users* **Objective**

This common guarantee has different assurance levels across the three implementations, reflecting our assessment of the difference in reliability of the three authentication procedures. The source of these differences is the

Objective-to-Policy correspondence, which in each of the three cases consists of a guarantee of the form

Policy *X* **satisfies the** *Authenticate Users* **Objective**

but which, like the overall guarantee, has different assurance levels in the three cases.

This construction is consistent with the view that, in deriving three different policies each of which targets the same system objective, the framers of the policies must have been using different procedures in order to produce three results of different quality. In particular, they must have used an approximate definition of ‘‘satisfies’’, and varied the fineness of the approximation. Depending on the details of the definitions of guarantee and assurance, this variation may be evinced by differences in the measures developed for the assurances assigned to the guarantees, or by differences in both the guarantee and assurance measures.

At this point, we are inspired to make a short digression. Figure 3.2 describes an organization of hierarchical specifications that allows different levels of compliance with a common objective, in the same way that the published national evaluation criteria describe different evaluation levels. Also, figure 3.2 mirrors the criteria in one other respect, by having all of the ‘‘looseness’’, or variability, concentrated in the Objective—Policy correspondence. This seems to match the currently predominant method of accommodating multiple levels of strictness or compliance, in which looseness in assurances is not considered desirable in the lower layers, but is actively encouraged in the Objective—Policy pairing.

As mentioned earlier in this report, the current thrust of research into security modelling includes an effort to extend formal assurance methods used in lower layers to include all levels in the design hierarchy. It is likely that methods that are successful in dealing with the strictly defined satisfaction of the lower layer guarantees may prove to be inadequate at the Objective—Policy boundary, where more flexible forms of satisfaction seem to be warranted. New methods will be required, perhaps ‘‘fuzzy’’ones, that permit limited agreement of guarantee policies to objectives.

In fact, security modellers have already encountered a version of this problem. Sometimes, the higher-level policies may require such strict compliance with the objective that some looseness in the Policy—Specification pair may seem desirable. For example, a policy that requires the elimination of all covert channels is generally regarded as being unenforceable in any usable system. We will not further consider organizations that have flexibility in the lower levels. Degrees of adequacy should be decided at the policy level, and visible to the policy makers, because we believe that doing otherwise will lead to problems with observability of and effective control and regulation of this flexibility.

3.3. GUARANTEE MEASUREMENT

Although we find the concepts of guarantee and assurance useful as defined, a difficulty is encountered when we try to fulfill the prescribed requirement that guarantee must be measurable. A guarantee is an assertion of the form **A satisfies B**, where **A** is a functional specification and **B** is a security policy. Since an assertion is not a quantity, and therefore is not itself a measurable thing, we must instead measure some aspect of the guarantee.

Which aspect we choose to measure depends greatly on what the measure is to be used for. No guidance is available here from the statement of work, which requires that a measure exist but gives no hint as to what it should reflect or what its purpose might be.

3.3.1. Measurable Aspects of Guarantee

The most likely expectation is that the measure should somehow reflect the ''quality'' or robustness of the guarantee. This could be, for example, a measure of the amount by which the functional specification could be loosened while maintaining the validity of the assertion. However, anything relating to validity is, according to the definitions, a matter of assurance rather than of guarantee, so it is not possible to construct a guarantee measure along these lines without confusing the boundary between the two.

Probably, a more acceptable measure of the ''quality'' of a guarantee would be the simple measure of the ''quality'' of its security policy **B**. The security functionality would be ignored by such a measure, which would then mean that no ''credit'' would be received for functionality in excess of that needed to satisfy the policy asserted by a guarantee. This might be justified by arguing that the test of whether or not such excess functionality is significant should be whether or not it permits the satisfaction of a higher quality policy.

One possible rebuttal of this is that, if policy grades are defined as discrete steps, as in existing national criteria, then enhancements which approach but do not reach the next recognized step will still get no credit even though the enhancements do allow the satisfaction of a higher-quality policy. The answer to this objection involves recognizing that a significant improvement in achieved policy may require a ''critical mass'' of improvements in security functionality, or it may require simultaneous and synergistic improvements in more than one area. In other words, there is a discontinuous jump in quality when some particular functionality boundary is crossed. In these cases, it is entirely correct to grant no extra credit for a job partly done. Since we have no mechanical method of distinguishing this case from that in which quality increases continuously with functionality, it is prudent to treat all cases according to the more conservative rules.

A second objection concerns the situation where one specification clearly exceeds another, but in a way that is not quantified in the policy, or for which the policy specifies only minimum values. For example, a security policy may require that a computer system support at least four security levels and eight categories. It may be that these numbers are not modified in the requirements for any higher class of secure system. In some applications, an implementation that supplies more than the minimum may be more desirable than one that just meets the criterion. It might, therefore, be appropriate to attach some notation to the measurement that provides this information.

It is important, however, that any such auxiliary measurement must be considered orthogonal to, and subsidiary to, the main guarantee measure. Failure to do so would make it possible for a system that meets only a low-grade policy to, by exceeding the specified minima by higher and higher amounts, masquerade as an arbitrarily high-grade system, which is clearly undesirable.

3.3.2. Relationship to Criteria Documents

Having reduced guarantee measurement to a question of policy measurement, we must now determine under what conditions we can make supportable assignments of measures to different policies. This is a question that, in various disguised forms, can cause great confusion, although at least part of an answer can be easily provided here.

Various government-sponsored evaluation guidelines, including the Canadian TCPEC, the U.S. TCSEC, and others, specify sets of attributes which distinguish different levels of trustworthiness of information processing systems. In the U.S. document, trustworthiness is not separated into its guarantee and assurance aspects, but in more recently published criteria a separation is made. The separation tends to be between assurance and functionality rather than assurance and guarantee, so a modest amount of translation is necessary to relate the contents to our framework.

Let us consider the functionality classes of the (May 1990 Draft) Harmonised Criteria of France, Germany, the Netherlands, and the United Kingdom. There are ten defined classes in all, numbered F1 to F10. The first five of these are intended to correspond to the functionality components of the US TCSEC classes C1, C2, B1, B2, and B3/A1 [1], respectively.

In the terms employed in this report, F1 through F10 represent different policies, and assertions of compliance with them constitute different guarantees. The functionalities required by classes F1 through F5 are the most

[1] In the US document, the levels B3 and A1 are functionally identical. A1 is distinguished by the greater use of formal methods in the design and verification process, which is reflected in a higher assurance rather than a different guarantee.

instructive for us to study. They are hierarchical, because the demands made at each level are a superset of the demands made at the next lower level. At the same time, there is an obvious unity of purpose among them that is not found among the other five classes. It is clear that F1 through F5 correspond to different ‘‘quality’’ levels, in the loose way we have been attempting to use that word [2], and so should be granted different measures of guarantee by any metric we propose.

The classes F6 through F10, on the other hand, do not share any similarity of purpose. They are patterned after classes defined by the German Information Security Agency for use in particular applications or environments which emphasize different aspects of security, and so are not directly comparable with one another. It is unclear how the guarantee metrics assigned to these classes should relate to one another, or indeed, whether it is even possible for them to do so in any meaningful way.

The notion of increasing quality that we wish to capture is tied to the increases in functionality required as systems progress through the classes from F1 to F5. Each new added requirement may increase the scope or granularity of some control or audit function, or it may add specific constraints in an area that was more loosely specified at lower levels.

For example, consider the access control requirements. At level F1, the nature of access rights are not specified, and it is not necessary that they be applied to all objects in the system. There is a requirement that it be possible to completely deny users or groups of users access to objects, and that only authorized users can grant or revoke rights.

At F2, there is the added requirement that rights with respect to an object must be specifiable for individual users, and that some form of "read-only" access be available. Also, mechanisms (of an unspecified nature) must exist to control the propagation of rights.

With F3, the nature of the access controls becomes much more closely specified. They must include mandatory controls, of the familiar clearance/classification type, that apply to all subjects and all storage objects. These must be based on a dominance relation involving hierarchical levels and non-hierarchical categories, and must prohibit read-up and write-down.

In F4, there must be support for the grouping of access rights to define roles. At a minimum, the roles of ‘‘operator’’ and ‘‘administrator’’ must be definable. The final level, F5, adds only a few extra wrinkles: it requires separation between the roles of operator, system administrator, and system security officer, and that the access control system give the functionality of a full access control list implementation.

---

[2] Without defining it, to be precise.

These functionality classes have the property that any system satisfying one criterion will also satisfy all higher-numbered criteria. From this observation, it is reasonable to determine that the minimum quality guaranteed by the higher-numbered classes is no lower than that of the lower-numbered ones, but it is not possible to conclude that they are of strictly higher quality, simply because they are more restrictive. That judgement would imply that the extra restrictions actually progress toward some desirable goal, and such a statement can only be made after identifying and considering that goal.

The only creditable method of measuring the policies F1 through F5, then, must involve considering their relationships to their common goal. Those relationships are precisely the Policy—Objective level guarantees and assurances. For the sake of consistency, it is probably best that we consider these guarantees to be equivalent, since they do have the same objective, in which case the measures of the Specification—Policy guarantees are related to the assurances of the Policy—Objective guarantees.

To generalize the above, allow the guarantee whose quality is being measured to be between any two levels N and $N-1$ in the specification hierarchy, where level 1 is the uppermost or Objective level. Let us call this a *level N guarantee*. The quality of this guarantee is the quality of its level N specification, which is the ‘‘target’’ of the assertion in the guarantee. Although we could define this quality to be a function of the level $N-1$ assurance, the resulting definition would only be useful for comparing among level N guarantees that correspond to the same level $N-2$ specification.

This is pictured in figure 3.3 for two systems derived from the same ultimate objective. The level 4 guarantees, that is the guarantees between the Specification and Detailed Design, are indicated in the figure. Beside each guarantee is shown a dotted line representing its corresponding assurance. If we attempt to use the level 3 assurances (marked with asterisks) to gauge the quality of the level 4 guarantees, we will end up with incommensurate measures, because the two level 3 assurances do not refer to the same level 2 Policy goal.

Instead, the utility of the measure may be increased by relating it to the assurance corresponding to the *cumulative* guarantee from level N to level 1. In figure 3.3, these cumulative guarantees and assurances are indicated for level 3. The cumulative level 3 guarantees both refer to the same objective at level 1, and so the cumulative level 3 assurances that are associated with them are comparable measures of the quality of the alternative level 3 specifications. In turn, this means they are also valid measures of the level 4 guarantees.

In general, then, we choose to measure the quality of a level N guarantee by the measure of the cumulative level $N-1$ assurance. Determining that cumulative assurance need not necessarily require knowing the cumulative guarantees, because it may be that the cumulative assurance can be calculated directly as the composition of the level 2 and level 3 non-cumulative assurances.

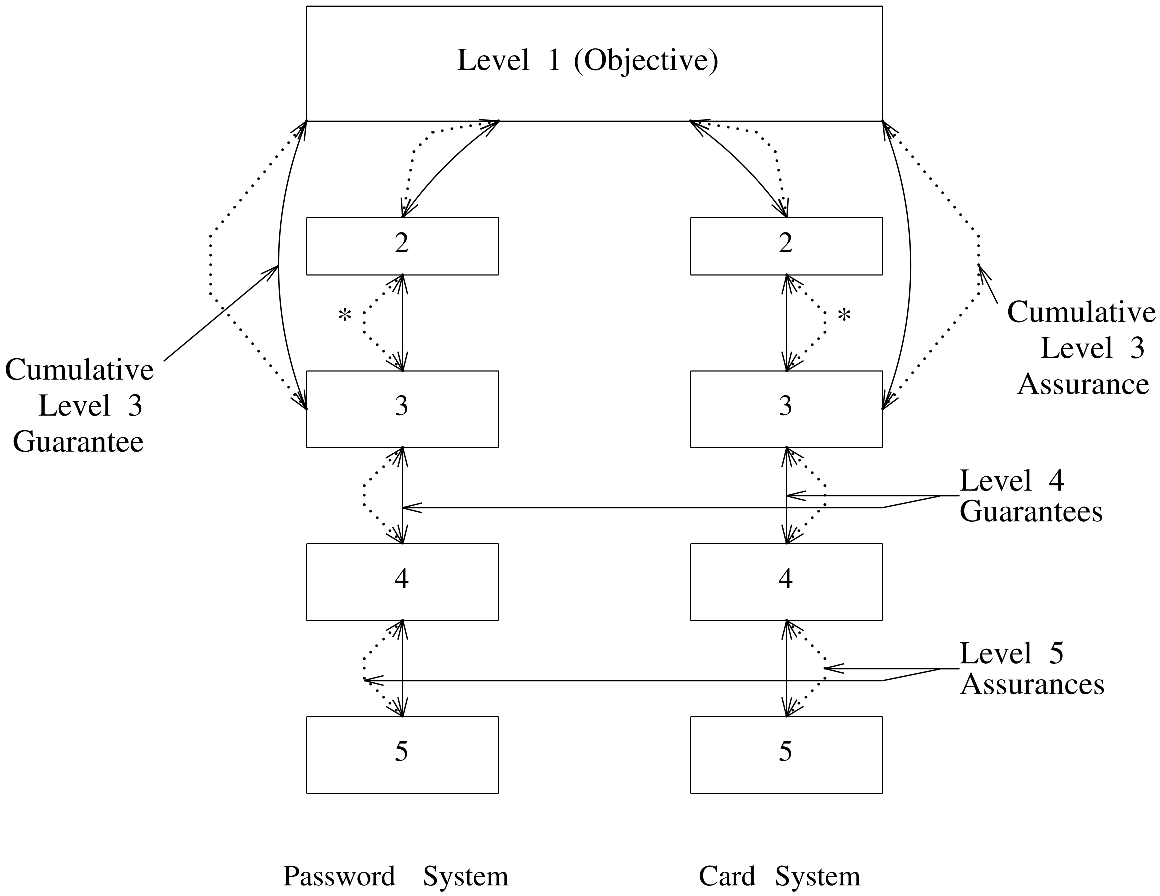


Figure 3.3: Guarantee and Cumulative Guarantee.

This alternative, however, does still require that assurances be determined for both levels 2 and 3, so that they may be composed to form the cumulative assurances. This may be difficult in cases where informal methods have been used at the higher design levels (notably, in the derivation of Policy from Objective). Similarly, when attempting to measure the quality of a level 3 or 4 guarantee, it may not be practical to accurately evaluate the assurance of an Objective—Policy mapping performed much earlier by a different group of individuals, and so it may only be possible to make a rough estimate of the assurance of that link in the chain. In that case, the accuracy of the derived guarantee measure may suffer, and so it may be preferable to use a measure determined by the cumulative assurance from level N to level M, where $M > 1$.

To accommodate this situation, we will use the convention that the unqualified term ''guarantee measure'' will refer to a measure taken relative to the guarantee's level 1 objectives, but that, when desirable, measures may also be taken relative to a common lower-level specification, given that one exists. In that case, the measure should be called a ''relative'' measure, and the basis specification should be identified, if possible.

A precise statement of the form of a guarantee measure must wait until assurance and assurance measures have been considered, but in the interim we will simply state that the measure of a guarantee should be that component of the assurance ranking that relates to how strictly and how reliably the specified policies or mechanisms will enforce the desired behaviour.

3.4. COMPOSITION OF GUARANTEE

Setting aside for a time the final resolution of guarantee measurement, it is appropriate to consider the question of guarantee composition, which can be stated as follows:

> *Given some number of components that are composed to form a larger system, what guarantees for the aggregate can be derived from the guarantees of the individual parts?*

A guarantee consists of a combination of a specification and a policy statement, where the statement describes some characteristic or property of the specification. That being the case, there obviously exists for each particular composed system the straightforward approach of tossing out the policy statements, combining the specifications in accordance with the specific composition, and then analysing the resulting aggregated specification to derive whatever new properties can be discovered. To disqualify this approach, which is exactly the type of labour-intensive full system analysis that we are striving to avoid, the problem statement is amended with the addition of

> *... without significant manipulation of the specifications?*

It is certain that nothing significant can be said in the most general case, that is, for arbitrary components, compositions, and policies (or, equivalently, properties of interest). However, it may be possible to make nontrivial statements if one or more of these is restricted in some way.

One simple case where this is true is when the property of interest is the weight of a system. Weight can be determined knowing only the weights of the components, and is independent of composition. Power consumption of a composite electrical system can be similarly derived knowing only the corresponding properties of the components, but requires the slightly stronger stipulation that some rudimentary rules of composition be followed, such as connecting power only to power inputs, and not shorting or wiring together outputs.

In the case of more complex properties, such as those relating to security, the answers do not come so easily. First it is necessary to understand the structure of systems, types of composition, and something of the nature of properties, before classes of each can be defined and investigated.

3.4.1. Abstract Systems

The study of *General Systems Theory* was inaugurated in the 1930s and '40s by Ludwig von Bertalanffy, a biologist, who was struck by the similarities in structure of problems arising in various fields and the corresponding similarities evident in their solutions. The now classical example is the analogy between a R-L-C electrical circuit and a mass-spring system with frictional losses, both of which, when disturbed, exhibit damped sinusoidal oscillations with exponentially decaying envelopes.

Von Bertalanffy suggested that the study of systems could be unified by abstracting away their identities, leaving only quantities and the interrelations that define their interactions. Consideration of these abstracted systems would lead to *the formulation and derivation of those general principles that are applicable to ''systems'' in general.* [3]

Major characteristics of the systems approach include recognizing the correspondences between similar systems in different areas that can be treated as a single abstract problem, and treating the abstracted system as an independently studied entity. Category theory, which has recently seen some use in formal security modelling, is a mathematical expression of these principles in which a system can be described by its variables or states, which form the *objects* of a category, and some relationships between them, the category's *morphisms* , and where the similarities between systems are expressed by mappings between categories, called *functors* . Formal manipulation of systems also invites contributions from graph theory, automata theory, and the theory of topologies.

Although the details of these formal models are not of interest here, we will take a brief look at the definitions and methods of classification employed in general systems theory, and consider the applicability of some of its results.

---

[3] [Bertalanffy68], pp. 32, 253

3.4.1.1. System Representation

When discussing the structure of a system, it is necessarily the structure of a model of the system that is being dealt with. And, inevitably, when the model is being conceived, some system aspects will be suppressed and others spotlighted. For any given system, the model chosen will depend on which set of characteristic quantities are under study.

Yet, independent of the system studied or of the system aspects described, a system can be considered as a collection of coordinated values of selected system attributes. In its most general form, this collection takes the form of a relation. The attributes may be simple values, in which case the model is a static one, or they may be time-dependent, in which case the model is dynamic. In addition, the attribute values, as well as the time values in the dynamic systems, may be considered either as discrete or continuous quantities. Of course, the same physical system may be represented by more than one kind of models for different purposes. For example, early in the design process a combinational logic device may be represented using a truth table or logic equations, yielding a discrete static model, while later on when signal propagation and transition times are considered it must be modelled as a continuous dynamic system.

In engineering use of systems there are usually some attributes which are externally controllable, and others which are only observable. This leads to the classification of attributes as *inputs* and *outputs* . The system description then takes the form of pairs of inputs and corresponding outputs.

It is also possible to introduce a generalized notion of a *state* as follows [after Mesarovic68]: assume the existence of an partial operator C (presumably for ‘‘Combine’’) which maps pairs of system behaviours into a combined behaviour. For example, the operands may represent system behaviour before and after time $t = 0$, and the combination of the two is a system behaviour defined over all time. The fact that this is a partial operator means that this combination is defined only for certain pairs; in the example, this would mean that not all $t > 0$ behaviours could necessarily follow all $t < 0$ behaviours. Now, if S is the set of behaviours, let Z be another set (the set of *states*) and define the relations $past-to-state \subseteq S \times Z$ and $state-to-future \subseteq Z \times S$ such that

$$C(s_1, s_2) \text{ is defined}$$

if and only if there exists some $z \in Z$ such that

$$(s_1, z) \in past-to-state \quad \textbf{and} \quad (z, s_2) \in state-to-future$$

That is, the state z encodes the ‘‘connection’’ between the past and future behaviours. Of particular interest are cases where $past-to-state$ is single-valued (i.e., the past yields a unique, well-defined state), and where no two s_2s corresponding to any given state z have the same inputs (i.e., outputs are deterministic).

Normally, the state of a system is not thought of quite so abstractly, but rather is identified with measurable values of changeable system attributes, much like inputs and outputs are. In fact, there is an overlap between the notions of inputs and outputs and that of state, and occasionally systems are modelled by lumping all these together as ''state'', particularly when dealing with systems that operate without any internal clocking mechanism. In clocked systems, however, the overall state is commonly divided into inputs, which are externally controlled values, the states proper, which are internally controlled, and the outputs, which may coincide with or be derived from the state values. Still, this division is made for the sake of convenience, and is not strictly necessary.

Although presented above as a connection between past and future, the general state can have other interpretations. In particular, if the combining operator represents an interconnection of components to form an overall system, then the ''states'' represent the restrictive effect that one component has on the possible behaviours of another. In this interpretation, a particular state is identified with a particular behaviour at the interface between two subsystems, whose individual behaviours are analogous to the past and future behaviours of the time system.

The conventional time-oriented notion of state gives rise to a dynamic representation of input-output systems, consisting of one function which maps a current state and remaining input pair to future outputs, and a second which maps them to a future state. The relationship between the dynamic and the behavioural representations, the latter typified by execution traces and input-output relations, is one that is of significant interest to practitioners.

Formally, at least, the two can be defined so as to be equivalent [Mesarovic68]. Practical concerns, such as conciseness and ease of manipulation, often suggest a state-based representation for the system, while design criteria and performance specifications are often more naturally expressed in behavioural terms. It is therefore often necessary to deal with both representations, and to convert between them.

System representations can also be distinguished by the nature of their inputs and outputs and state variables, such as whether they are continuous or discrete, ordered or unordered.

3.4.1.2. System Structure

Structure is present in many forms in systems. The most familiar of these is the structure of interconnection between components in a complex system. Each component is self-contained, and can be described in one of the ways previously enumerated. Components interact with their neighbours via shared variables that may be represented either as coincident inputs and outputs, or as overlapping state components.

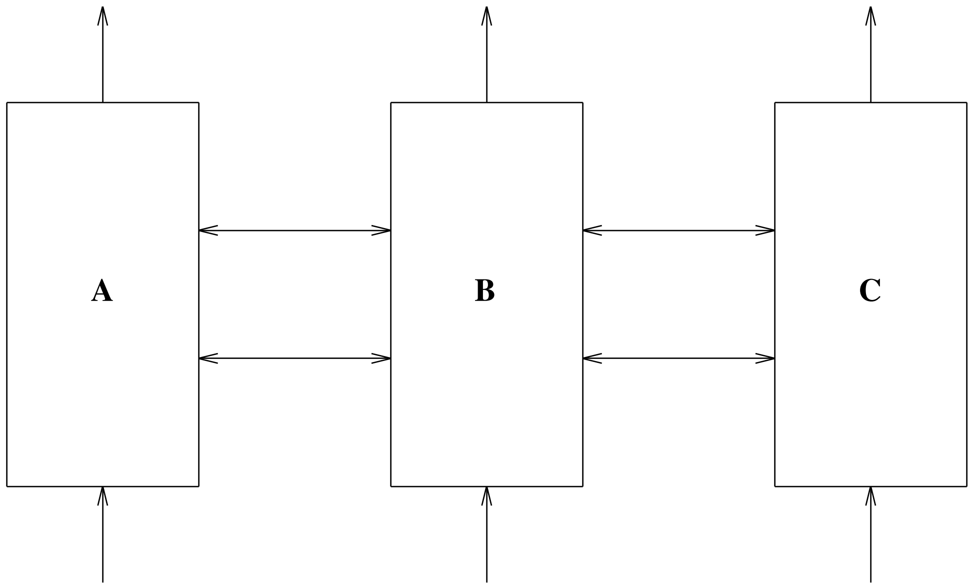


Figure 3.4: Parallel Composition.

When interconnection structure is represented in graphical form, the shape of the graph often reflects the flow of information and control in the composite system. Components may be connected in a parallel structure, in which they each receive inputs from the same kinds of sources and deliver outputs to similar destinations, as in figure 3.4. This structure is commonly encountered in networks of autonomous systems. There, the composed systems are often roughly equivalent in functionality and in the various policies they enforce, and their communications can be characterized as peer interactions. It may be that the composite system behaves very much like one of its components, but with an expanded capacity.

Serial interconnection, as in figure 3.5, is a configuration that tends to be used with a different class of components. The structure may reflect a layered design, in which each layer adds some functionality or provides some service different from that supplied by the other layers. Various degrees of interdependence between the layers are possible. The transport and network layers of a networking protocol implementation are an example where one layer uses the services provided by the layer beneath, and so there is a definite functional dependence between them. However, one could add to this an encryption layer, or a usage accounting layer, or both, so that they would add their respective functionalities without depending on the existence of the protocol layers beneath.

In these and other structures, one may distinguish the subset of ''loopless'' structures. Loopless structures are generally easier to deal with than ones containing loops, because there is no need to identify behavioural fixed points. A consequence is that the stability of the composite follows directly from the stabilities of its components. This is not necessarily the case for structures with feedback, where stable components may be interconnected to form unstable composites.

A related phenomenon is the non-composability of some nondisclosure properties [McCullough87]. These typically involve a component that may enter an unsafe state in response to some high input, but which will spontaneously decay to a safe state. If this decay is marked by some externally visible event, such as the emission of a high-level output, then feedback can be used to force the component back into an unsafe state, keeping it almost continuously unsafe. Loopless networks do not display this failure mode.

In any structural system, care must be taken to properly identify the significant components. A system like the one modelled in figure 3.6 (a) may actually be an instance of figure 3.6 (b), with a nontrivial interconnect component such as a local area network. Viewing this system as an instance of (a) may be dangerous, since it assumes that the interconnect component functions so that traffic on each of the three paths is always routed correctly and cannot interfere with traffic on other connections. If these assumptions may fail to hold, there is a potential to overlook system functional failures and

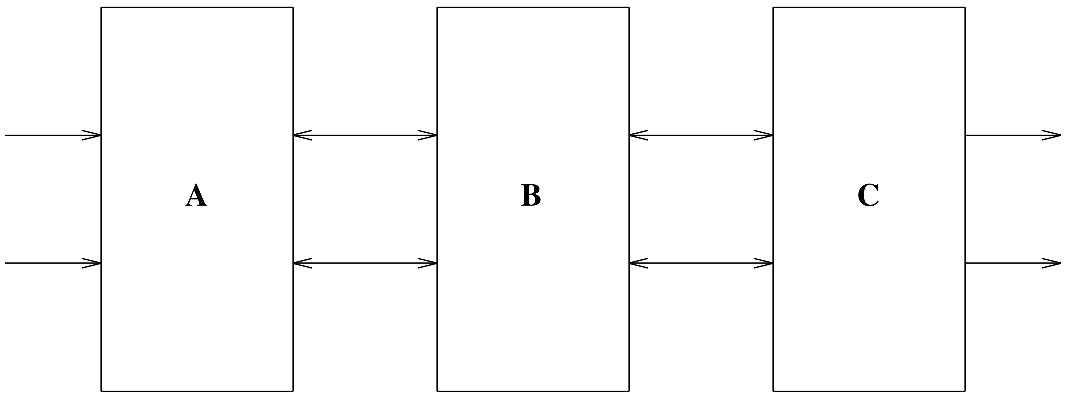


Figure 3.5: Serial Composition.

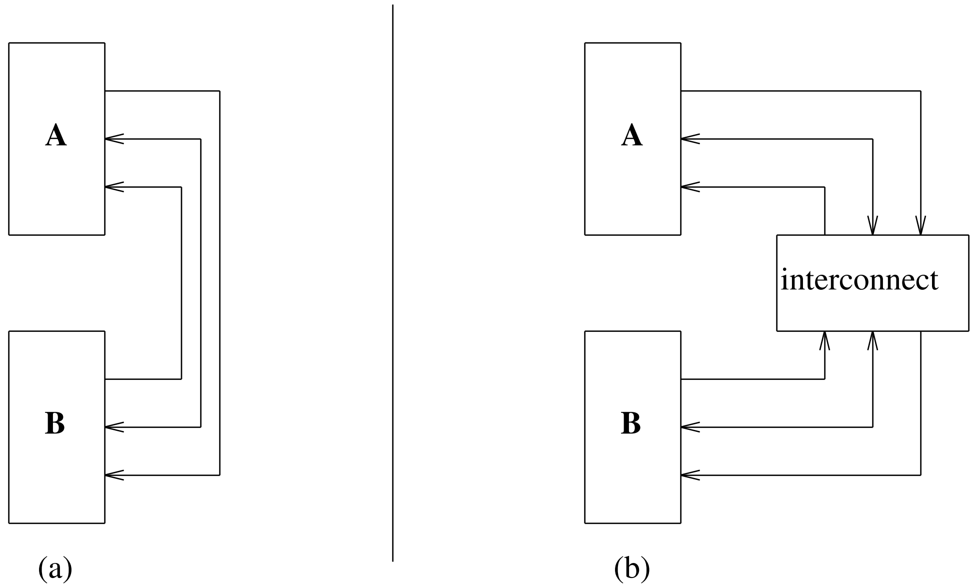


Figure 3.6: Connections as Components.

violations of policy.

Less familiar than interconnection structure is the idea of structure in state-space. A graph of the system states and transitions can be explored for structural features such as connectivity or the existence of cliques or minimally connected subsets. Additionally, the states may be factorable, allowing a complex transition function to be represented as the product of a number of simpler functions. These kinds of structure do not necessarily bear any relation to the physical composition of the modelled system. They are most useful when analyzing the state space of a complex system, perhaps with a view to decomposing it into smaller factors, as might occur during the design process.

3.4.1.3. System Properties

‘‘Property’’ is a term that is often used freely and without any formal definition, usually with an intended meaning resembling ‘‘a system characteristic''. This notion of ''property'' is a highly inclusive one, and can be extended straightforwardly to comprehend any statement about a system that we might care to make. Such generality is as much an obstacle here as it is with regard to other system and composition concepts, and so we are lead to consider classification of properties into more restrictive categories.

Static properties are the simplest sort of property. In the field of security they are usually only of interest when dealing with passive articles, such as the opacity of an envelope or the weight of a safe.

From the viewpoint of data processing security, the more interesting properties tend to be behavioural in nature. These take the form of statements about the behaviours exhibited by a system. Satisfaction of the property may require that the statements be true of each system behaviour, or only of some of them. Alternatively, the property may be a statement about a collection of behaviours and their relationships to one another, rather than a test applied to individual behaviours.

A number of researchers have adopted the former notion of property in their formal definitions [Lamport84] [Alpern85] [Abadi90]. They consider a *property* to be simply a set of behaviours, possibly with the added requirement that the set also include all variations on its behaviours generated by finite repetitions of states ("stuttering-closure"). This type of property, which could be termed a *simple property* (S *–property* in the following), is essentially a test performed on individual component behaviours: a behaviour satisfies the property if it is contained in the corresponding collection. Also, a component is said to *have* a given S-property if all of the behaviours the component can exhibit are contained in the property.

Two major subclasses of S-properties, the *safety* and *liveness* properties, have been distinguished. A safety property is essentially an S-property stipulating that some undesired thing does not occur during execution. Common examples and their corresponding undesired things include

mutual exclusion
: Two or more processes executing in a critical region at the same time

partial correctness
: Termination without satisfying the applicable postcondition

nondisclosure
: Delivering sensitive information to someone who is not authorized to receive it

Liveness, on the other hand, is a stipulation that some desirable thing eventually does happen. Examples are

program termination
: Executing the final instruction

non-starvation
: Access to a shared resource is eventually provided infinitely often

Sometimes the classification of a property as safety or liveness can be difficult, and intuition may be misleading. Apparently very similar properties may be classified oppositely to one another. Many people would probably consider ''freedom from deadlock'' to be a liveness property, because it implies that the system is always ''doing something''. In fact, it is a safety property. The undesired behaviour in this case is the system's entering a deadlock state, which prevents it from progressing. And yet, the closely related ''freedom from denial of service'' *is* a liveness property. Its desired behaviour is the eventual servicing of every request.

It might appear that making this distinction is merely lawyering with definitions, and to some extent this is a valid criticism. Although freedom from deadlock and freedom from denial of service are not identical properties (it is possible to deny service without involving deadlock), one could define a property by the stipulation ''eventually enters a non-deadlock state'', that is a liveness property and yet is difficult to distinguish from the deadlock-free property [4]. Confusion can be created by transforming a safety property *X never happens* into a form of liveness *not-X always happens* , or perhaps *not-X always eventually happens* , and this seems to be the root of the problem. Yet, in formal analysis the two types of properties are verified using different methods: safety by arguing the preservation of an invariant, and liveness by means of appeals to the exhaustion of finite sets. This implies, at least, the existence of some fundamental difference between them. Originally, safety and liveness were informally recognized as being of different character, so there is also evidence of the intuitive recognition of a distinction.

[Lamport84] first formalized the distinguishing characteristic of a safety property by reasoning in this way: If a safety property P fails to hold for some execution, then at some point some undesirable occurrence must have happened. P says that this bad thing must never happen, therefore once it has happened, it cannot be ''erased'' from the past, so it must be the case that the situation is irremediable. So, no execution prefix that is not in P can be extended to an execution in P. Equivalently, a property P is a safety property iff, for any behaviour b that does satisfies it, all finite prefixes of b also satisfy it. This definition restricts the kinds of undesirables we can talk about to those that can be definitively said to have occurred by a particular point in an execution.

[Alpern85] completed the set by listing three alternate precise definitions of liveness. The one its authors prefer is based on the observation that, if a bad thing has not happened up to a certain point, it is always possible for it to happen in the future, so failure to be live *is* remediable. In other words, for every finite behaviour prefix p, there exists at least one conceivable extension of p that satisfies a liveness property Q. This definition does not require

[4] That they are still not precisely equivalent is apparent from the fact that an execution that does nothing is deadlock-free but does not satisfy the liveness property.

that the desired behaviour be a single event, or even a finite number of events, although these are certainly permitted. Alternatively, it can require that an occurrence happen infinitely often. Liveness cannot require that something happens all the time — that requires a safety restriction prohibiting noncompliant behaviour — but can specify that it will always happen at least one more time.

With these clearer definitions it is possible to resolve the confusion encountered earlier. *Not-X always happens* does not describe a valid liveness property, and although *not-X always eventually happens* is one, it does not absolutely prohibit X, and so is not equivalent to the safety property *X never happens* .

The only remaining confusion results when the semantics of X, the prohibited event, are such that it encodes information about future execution that is itself liveness related. For example, if X is defined to be *entering a state from which Y cannot happen* , then the apparent safety property *X never happens* is equivalent to the liveness property *Y always eventually happens* . In essence, this amounts to labelling the violation of liveness an event, and then prohibiting that ‘‘event’’. The feature that distinguishes this example from the others is that in order to recognize that X has occurred at some point in an execution, it is necessary to consider all possible future extensions of that execution. Thus, the occurrence of X is not a statement about what has already happened, but an assertion about future behaviour. Careful consideration of this leads to the conclusion that the original property is not a true safety property, because it is not irremediable. There is no finite execution prefix that violates it ‘‘once and for all’’, since any prefix can be extended with one or more Ys, nullifying the violation [5].

In addition to these important types of properties that are defined for individual behaviours, there are properties which can only be defined for a collection of executions. Determinism, characterized as an assertion that no two distinct executions may have the same subsequence of inputs (and same initial state, if applicable), is one such property. Also in this category are closure statements along the lines of

> *If the system can exhibit behaviour* **A**, *then it must also exhibit the related behaviour* **A′**

More than two executions of a system may also be related by one of these types of properties. The non-deducibility definition of confidentiality [Sutherland86] requires closure with respect to a function that merges two executions to form a third. These are not S-properties in the sense defined by Lamport or Alpern, because they are attributes of a collection of behaviours rather than attributes of the individual behaviours themselves. These

[5] Irremediability must hold for arbitrary extensions, not just for extensions satisfying the system specification.

collection-oriented properties (which we will distinguish as *C−properties*) are not preserved by arbitrary subsetting as S-properties are, and so that, in general, C-properties must be verified for the particular set of behaviours that an implementation exhibits rather than a more conveniently defined superset.

A surprisingly complex aspect of system properties relates to the conditions, either explicit or implicit, under which a property holds for a given system. There are few properties that could be depended on to hold in an truly arbitrary environment, but the task of identifying and characterizing all the environmental assumptions that must be met before a property can be expected to hold is not a trivial one. For a complex system, in addition to the usual set of ambient temperature and humidity restrictions, *g*-force limits, electrical or other supply ranges, radiation levels, and the like, there are often many manuals full of procedures for installation, interconnection, setup, maintenance, administration, and operation, all of which are expected to be followed if the properties specified for the system are to be achieved.

3.4.2. Composition of Components

The principal question of this section concerns the composition of components, and the properties of the resulting composition. As explained earlier, is our intention to avoid the obvious method which involves a detailed functional analysis of the composed system. However, we recognize that in the general case of unrestricted properties, components, and interconnections, it is highly unlikely that any other approach would yield significant results. A consequence is that we must consider sets of controlled, restricted properties, components, and compositions. Identifying appropriate sets of these must be our immediate goal.

There are a number of modes through which the properties of the components of a composite are determined, as well as the properties of the composed system considered as a whole. The simplest case is when properties of individual components are directly exhibited by the composite system. In this case, the points at issue are whether the component properties continue to hold, and whether they are visible as properties of the composite.

1: *Maintenance of individual component properties*

> Do the properties satisfied by individual components continue to be valid in a composed system? Perhaps not, if interconnection has the effect of violating some of the environmental assumptions on which the original properties were based.
>
> If the assumptions are clearly stated, it is just as clearly the responsibility of the interconnecter to verify them. Implicit assumptions, on the other hand, may be unexpectedly violated when a component is used in an unforeseen application, or in a novel configuration. A communications line may be specified to operate at a high baud rate,

with the implicit assumption that there is a human typist at the other end who can supply input only at a much lower character rate, but may be overloaded if a machine sources data to it. Or, a nondeducibility-secure computer may be connected in a configuration with feedback, violating the implicit assumption that its inputs are independent of its outputs, and nullifying its security property.

2: *Visibility of component property as a system property*

In some cases, there may be no real difference between the property defined for the component alone, and the property defined for the component in the composite system. This is often the case when the system is basically only an aggregation of components, which continue to function more-or-less independently. Thus, a property relating the X-inputs and Y-outputs of a component may continue to hold in the composite, if the system's X-inputs all go to that component and its Y-outputs all originate there.

However, it may also happen that a property entails a relationship among component attributes that do not maintain some essential aspect of their identity during composition. ''X-inputs to the system'' may describe a different set of signals than ''X-inputs to the component''. Or, a component may enforce an N-person rule, where authorization from several users is required to generate a certain command output, but in the composed system a similar output can be fabricated elsewhere and substituted for it. Any composition in which there is a path around a component may exhibit similar behaviour.

A variation of the preceding is where a property shared by all components leads to the same property being visible in the whole. With respect to (simple) behavioural properties, there is no longer the danger that inputs may be routed around a compliant component and through a noncompliant one. In terms of the formalism used in [Abadi90], S-properties are preserved by composition. This follows because composition is modelled by intersection of component execution sets in [Abadi90], and intersection is really just a restricted form of subsetting, which does preserve S-properties. Violation of environmental assumptions is still a possible source of failure in this situation, however.

Difficulties may also be encountered if the shared property is a C-property. They are, generally speaking, not preserved by intersection, so that two components which each satisfy a C-property will not necessarily continue to do so when interconnected. ''Necessarily'' is a significant modifier, because there do exist C-properties that are conserved. It is known that the successful composition of components, each of which satisfies a safety property, results in a system which satisfies a safety property that is the intersection of the component properties. Since ''safety-propertyness'' is a closure requirement, this is an example of a preserved C-property.

Similar to the scenario of guarantees that are visible after composition is the situation in which components achieve distinct policy goals, and contribute these properties to the composite, but where there is mutual interdependence among the properties. In a trusted system, for example, a MAC enforcing component may depend on the proper functioning of a sibling encryption/decryption unit. Meanwhile, the guarantees that DAC and Audit components provide are meaningless without proper Identification and Authentication.

One way that this type of situation can arise is when a set of components are intentionally designed to be composable in this way. In such a case, they would be constructed to employ compatible protocols for passing identifiers and labels, and to supply appropriate support facilities to one another as required by their property interdependencies. They would be intended as building blocks, to be interconnected as required per installation, probably subject to some interconnection and structuring guidelines.

It is likely that the guarantee of each of these components would be expressed conditionally, making explicit the need for the functional support of the other components it is designed to work with. Satisfying the conditions is then a matter of satisfying the environmental assumptions made when specifying the components, a problem which has been observed earlier. If the set of dependencies contains no cycles when viewed as a directed graph, then the properties should aggregate as they were designed to. However, if cycles appear, the validity of the composition must be established, which requires more knowledge of the nature of the properties involved. [Abadi90] shows, for example, that cyclic dependencies on safety properties of other components do compose properly, but dependencies on liveness may not.

A new difficulty is injected if the interdependence is not preplanned. Beyond the obvious problems of interface and protocol compatibility, it is likely that the component guarantees have not been expressed so that the composite guarantee is a simple addition of the component ones. If a MAC component guarantees to enforce a MAC policy with respect to its identified users, and an Identification and Authentication component guarantees to supply reliable identification, we would like to conclude that a composition of the two will enforce MAC with respect to the true user identities. Even this minimal amount of guarantee synthesis is not a simple problem, if the guarantees are not phrased so as to make the dependence visible, and if the necessary interconnection rules have not been thought out in advance.

The most complicated cases of composition involve different properties of components combining to synthesize entirely new properties of the whole. In considering these synthesized properties, it is possible to distinguish those that are *emergent* , meaning that they arise from the composition itself and are of a different nature than the properties of components observed individually, and the non-emergent variety that are a simple combination or addition of component properties. By this measure, *weight* would certainly qualify as a non-emergent property, though *reliability* may also be held to be non-emergent, since the reliability of a composite system may be calculated by straightforwardly combining the reliabilities of its components.

It seems that almost all of the properties of a composed system that are of potential interest are emergent, at some level of analysis. A radio has the property that it receives a particular band of broadcast signals, while an automobile has the property that it carries passengers over distances. In both cases, it is easy to subdivide the system into components such that none of them display the corresponding property. Separating the radio into ''battery'' and ''rest of radio'' is one obvious decomposition. It is also one of the better behaved types of decomposition, because of the unidirectionality of the functional dependence between these components.

If we have the foresight to specify the property of the ''rest of radio'' conditionally, eg. *receives broadcast signals if connected to battery* , then it is relatively simple to determine the characteristics of the composite. If we lack this foresight, or if there exists a mutual interdependence between the components, then their interactions may be much harder to understand or to predict, given only the statements of the properties and not the detailed internal structures of the components.

Clearly, then, there are situations in which the synthesis of emergent properties is a complex process, and cannot be adequately described without considering the specifications of the components in greater detail than we wish. In many cases, these difficult composition issues may be avoided by restricting the types of composed systems or the types of properties considered.

The types of combining properties that might be expected to be amenable to less detailed analysis are those, like weight or power consumption, that are simple aggregations of similar component properties, ones that are combinations of independent properties, and ones in which there is a simple loopless dependency relation among the properties. Composite systems that display complex properties are more likely to consent to analysis if the components are relatively few in number and are themselves complex independent functional units, than if the analysis is conducted at a more detailed level.

4 ASSURANCE MEASUREMENT AND COMPOSITION

Incertainties now crown themselves assured.
William Shakespeare,
Sonnet 107,

Mad world! mad kings! mad composition!
William Shakespeare,
King John,
II, i, 561.

4.1. INTRODUCTION

In the previous chapter, we addressed the issue of *guarantee* while in this chapter we will consider *assurance* . Before starting, it is pertinent to review the meaning of these terms.

The statement of work issued for this research contained definitions of these terms, and required investigation of the concepts embodied in the definitions. A guarantee was defined as

> an assertion that a given security functionality is sufficient to satisfy the requirements of a specified security policy

while assurance was

> the level of confidence that a guarantee associated with a specified security policy is both valid and effective and that assurance is associated with both the quality and durability of the guarantee

In these definitions, the term *security policy* is considered to designate

> the set of laws, rules, and practices defining the protection of a system and the information it contains from unauthorized disclosure, modification, destruction, or removal and protection from the interruption of services that the system provides

4.2. ASSURANCE MEASURES

Assurance is a term commonly used to describe a level of confidence that some system will do whatever it is supposed to do. The details of the definition vary from one source to another, as various aspects of security are included under different headings.

The 1985 TCSEC describes assurance in these words:

> The third basic control objective is concerned with guaranteeing or providing confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy.

It then divides assurance into two types, life-cycle assurance and operational assurance. Life-cycle assurance involves the use of controls, standards, and procedures during the design, development, and maintenance of a system, to ensure that it is constructed so as to provide an accurate and sound interpretation of the security policy. Operational assurance is concerned with the features and procedures which ensure that the policy enforcement features are active and not circumventable during system operation.

The 1987 version[TNI87] of the Trusted Network Interpretation of the TCSEC (TNI) contains a similar definition of assurance as the

> ... basis for believing that the functionality will be achieved; it includes tamper resistance, verifiability, and resistance against circumvention or bypass.

The draft[CTPEC90] of the Canadian Trusted Computing Product Evaluation Criteria (CTCPEC) adopts essentially the same definitions as the TCSEC, and differs from that document primarily in the unbundling of assurance from policy in its rating system.

The first version of the Green Book[ITSEC89] takes a slightly different approach, and defines its version of assurance in this rather roundabout way:

> The security functions selected to satisfy the security objectives form but one aspect of the technical security of a product or system. No less important is assurance that the selected functions will indeed satisfy the security objectives.
>
> Assurance needs to be addressed from several different points of view and, in these harmonised criteria, it has been decided to distinguish confidence in the correctness from confidence in the effectiveness of the security functions and mechanisms.

The part of the ITSEC dealing with confidence in correctness is the counterpart of the life-cycle and operational assurance of the TCSEC. Assurance of effectiveness is a new addition:

> Evaluation of effectiveness assesses whether the security functions and mechanisms that are provided in the [target of evaluation] actually satisfy the security requirements.

This evaluation concerns five points:

1. Suitability of the implemented functionality

2. Binding (really, mutual compatibility) of the security functions
3. Consequences of vulnerabilities
4. Ease of use.
5. Strength of mechanisms against direct attack.

In all of these definitions, there is a common thread of confidence in the validity of a correspondence. The identities of the correspondents distinguishes the variations: Correspondence between specification and design, between design and implementation, and between specification and intention. In one way or another, these are all contrasts between ''theory'' and ''practice'', or, equivalently, between an ideal and successively more realistic approximations to that ideal.

The approach taken in this report involves making the following generalization of this correspondence: Assurance is the basis for believing that a given *implementation* will achieve a specified *goal*.

When the goal is a policy and the implementation a specification, it is the reliability of the high-level design process that is being assessed. When the goal is a specification and the implementation a detailed design, we enter a domain that includes activities like hardware and software verification and correctness by construction. These two levels correspond to the ''assurance of correctness'' found in the various criteria.

When the goal is a policy objective and the implementation a specific policy, we are talking about confidence in the policy design. This is a form of ''assurance of effectiveness'' that is not encountered in the criteria documents. Those documents treat only a range of the design hierarchy, accepting policy as the ultimate target, and do not take into consideration that policy also is an expression of a more abstract objective.

Finally, the physical realization and operation constitute an implementation with respect to that detailed design, and the assurance-related issues included here can be extended to encompass the efficacy of quality control mechanisms active during and after manufacturing, plus that of the distribution, configuration management, and operating procedures responsible for maintaining the effectiveness of the implemented policy enforcement mechanisms. This latter element is provided in recognition that an assurance rating must consider not only initial conformance of the implementation with its objective, but also the level of confidence that the correspondence will be maintained over the lifetime of the system.

4.2.1. Form and Significance of an Assurance Measure

It is clear that a measure of assurance should reflect a ''level of confidence'', but not quite as clear just how it should do this. The TCSEC lumped all issues of functionality and assurance together in its rating system, which is understandable from the viewpoint of providing guidance about the levels of effort in various areas of security that will properly complement one another, but which does not provide useful points of reference for assurance measurement.

Even those criteria documents like the that separately rate functionality and assurance do not employ direct measures for assurance as much as they specify benchmarks for assurance generation. That is, they prescribe procedures and facilities which are considered sufficient to achieve some level of assurance, without actually defining a measure for the assurance.

The ITSEC document is concerned more with assurance matters than with functionality. It defines six assurance levels each of which specifies procedures and levels of effort in the development process, the development environment, and the operational environment. Rather than inspecting this large document in detail, we will instead look at the somewhat less complicated CTCPEC and TNI assurance provisions, perhaps with mention of the ways that ITSEC varies from them.

The CTCPEC contains assurance levels composed of three independent ratings of seven levels each. The three axes, and the specific assurance concerns that comprise them, are extracted from the assurance components of the TCSEC ratings D through A1. However, the levels are additionally given the following names, which serve as clues to the nature of the measurement that they imply:

Level	Title
6 (highest)	Formal Design
5	Rigorous Design & Security Domains
4	Structured Design
3	Independently Assured
2	Independently Tested
1	Vendor Assured
0 (lowest)	Minimal

It is notable here that even the names of the assurance levels do not really refer to levels of assurance *per se*, but appear to relate to the level of effort expended in generating the assurance.

An alternative approach is that taken in Part II of the Trusted Network Interpretation (TNI), which deals with security issues in distributed systems that are not patently mappable into the classical single-system treatment of the TCSEC. Included here are general networking security elements, such as encryption and protocols, as well as specific network security services similar to those identified in the OSI Security Architecture [ISO7498-2].

In Part II, each service is independently rated according to its functionality, strength of mechanism, and assurance. The identified assurance levels are *none* , *minimal* , *fair* , and *good*, although there is no real quantitative information available concerning the significance of the different levels. The document does list various activities that contribute to assurance, but the only guidance given regarding what level goes with which activity are these parallels drawn between the Part II assurance levels and TCSEC classifications:

Minimal	$\Leftrightarrow$	C1
Fair	$\Leftrightarrow$	C2
Good	$\Leftrightarrow$	B2

Instead of this implicit method of specifying assurance, it may be desirable to consider measuring it in a more direct fashion. Our eventual goal is to combine assurance measurements, first across abstraction levels in a single design hierarchy, then among components in a composed system. Procedures useful in generating high assurance at one abstraction level, or in one system component, may be much less effective, if appropriate at all, when applied elsewhere.

This, in turn, implies that assurance ratings based on lists of procedures followed and sophisticated techniques employed will not easily be comparable, never mind composable, from one application to another. These efforts would be facilitated by a rating system that represents assurance directly, independent of the details of its origin.

The foregoing does not imply that we will necessarily strive to reduce an assurance rating to a single number. Design, operation, and life-cycle assurance are three independent axes of assurance, in the sense that overbuilding one cannot compensate for deficiencies in the others. We would expect a useful assurance measure to have at least this amount of structure.

Within each category, according to the fundamental definitions adopted in this work, an assurance measurement should reflect a level of confidence in the achievement of some specified guarantee. Probabilities are the first alternative brought to mind by this requirement.

In its simplest form, a probabilistic assurance measure for a guarantee relating two abstraction levels in the implementation hierarchy would indicate the likelihood that the derived, more concrete design is faithful to the more abstract. A 99% assurance level, for instance, would indicate that a similar design and validation exercise, using the same tools and procedures with similar apparent success, would actually be flawed on average one time in a hundred.

There are a number of reasons why simple numeric probabilities may fail to be appropriate for this task. First is the difficulty of calibrating the assurance-generating activities in order that meaningful probabilities can be assigned. In some cases it may be possible to do this quite accurately, as in the post-production testing of integrated circuits. By modelling the mechanisms that generate fabrication flaws, it is possible to estimate what percentage will be caught by any proposed series of functional tests. A compromise is reached between the complexity and duration of the testing and the reliability of its result, to achieve an acceptable post-screening failure rate.

In other areas, however, the failure modes are less well understood. They may vary widely depending on the habits of individual designers, or for different classes of components. It can be very difficult to estimate the frequency or consequences of design errors, as well as the reliability of different programs of testing, design review, or even formal verification.

In the case of security properties, where the likelihood of the existence of flaws is less important than the likelihood of their being exploited, the situation is further complicated by being dependent on the opponent's level of motivation and resources available. These difficulties are probably responsible for the lack of quantitative assurance measures in the criteria, and in the absence of applicable empirical or theoretical estimates they will continue to hinder the quantification of assurance.

Yet, even if these kinds of quantitative assurance data were available, it would still not be clear that simple probabilities of correctness would be adequate descriptions of assurance. In order to use a failure probability to accurately estimate risks, it is also necessary to know the mode and magnitude of the contemplated failure. A system with a guaranteed response time of 2 seconds, but an associated assurance of only 50 percent, might be acceptable if its worst-case were 2.5 seconds, but not if it were 2 minutes. In this case, a number of time/probability pairs, or a graph of the probability distribution, would be a much more complete and useful description.

Composite systems also introduce new complications to the guarantee and assurance problem. Consider a trusted system consisting of two similar components, one of which handles data of sensitivities A and B, and the other handling sensitivities C and D. The components themselves do not have a high assurance of data separation, but their composition is accomplished in a manner that very reliably separates data in the two components. Now, a simple guarantee of the separation of levels can only be given with low assurance, even though A and B level data is separated with high assurance from C and D. This situation could be represented more accurately if the assurance was given as a number of probabilities of different failure modes: A-B mixing being perhaps 5%, but A-C only .1%.

Furthermore, it is not usually the existence of a fault that is of prime importance, but the likelihood that a fault will actually affect the operation of the system in a deleterious way. In the case of security, where it is assumed that users or outsiders are deliberately trying to subvert the system's security properties, the fundamental issue is the user's ability to exploit a weakness, and this includes both the level of effort required to do so and the payoff it would generate.

This may be best illustrated by considering cryptographic systems, all of which contain at least one ‘‘bug’’:they offer no security if the key or key seed is discovered by exhaustive search, or indeed if it is guessed. Yet, highly assured cryptography is possible, if the rating authority can generate a high level of confidence that other weaknesses do not exist, and if the level of effort (or luck) necessary to exploit these is made sufficiently high [1].

The most valuable form of assurance rating, then, would be one that took into account all of these factors. It would allow the accurate estimation of the probability of guarantee violation, taking into account the magnitude of the violation, perhaps which part of a complex guarantee it involves, and where appropriate, the level of effort involved in inducing or exploiting it. A risk assessment procedure could then use these probabilities, after taking into account environmental factors such as external security that may affect the probabilities of some failure modes, or criticality information that would allow a particular failure (i.e., ‘‘may allow level 1 data to flow to level 2 containers’’) to be mapped into terms that are more mission-specific terms (‘‘may allow Secret data to flow to Confidential containers’’).

An ideal assurance rating system would then give us

$$Prob_{violation}\ (A, G, G', e)$$

where A is the collection of assurance-generating rules and procedures that have been employed, G is a possibly complex guarantee, G' is a corrupted guarantee representing the failure scenario, and e is some measure of expendable effort on the part of an adversary.

It is appropriate at this time to examine the precise form of guarantee failure that is under consideration here. The guarantee is an assertion of correspondence between one level of specification and its target, a more abstract specification. In figure 4.1, the first diagram shows the original guarantee G existing between two specification levels. In that case, the design and any subsequent verification have produced the expected faithful correspondence between them.

In the second diagram, the design process did not proceed correctly, and the implemented system does not represent a faithful transcription of the specification. Instead, the correspondence G' that was actually achieved links the existing implementation to a slightly modified upper level specification. This view of a corrupt guarantee highlights the distinction between the higher-level specification that the system was supposed to meet and the one that it actually does meet, and is the form of perturbed or

[1] The TNI takes a slightly different tack, treating security as a combination of functionality, assurance, and a separate third issue they call *strength of mechanism*. There, level of effort considerations would be reflected in a strength of mechanism rating. Since we have available only the rubrics of functionality and assurance, we must reclassify it as one of those, and it seems to fit more naturally into assurance than into functionality.

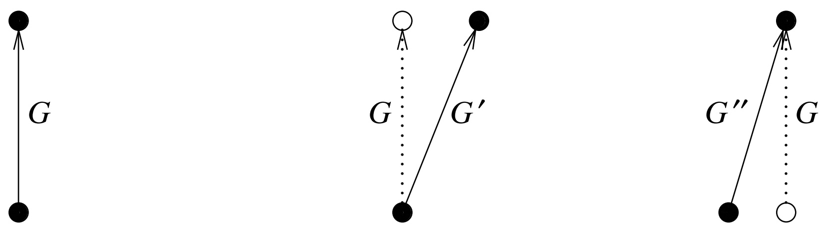


Figure 4.1: Modes of Guarantee Failure.

corrupted guarantee that is contemplated in the definition of assurance presented above.

The third diagram represents a possible alternative view of a this process, in which the focus is on the disparity between two alternative implementations. One is the guarantee G that is claimed to hold between the implemented system and the target specification, and the second G'' is the valid correspondence between a different (correct) implementation and the target specification. Our definition of assurance does not utilize this conception of guarantee corruption, because it does not give any indication of the extent to which the system that was actually implemented violates policy. Without such an indication, it is not possible to assess the risk that would result from using a system that displayed this type of failure.

Returning once again to our discussion of the probabilistic form of an ideal assurance measure, it must be admitted that the likelihood of being able to achieve such a measure, for any but the simplest of examples, is low. The problem is the difficulty of assigning accurate probabilities for assurance measures over arbitrary failures and adversary effort levels. Even for specific failures, the number of different issues affecting assurance, and therefore the number of calibration studies that would have to be done in order to be able to generate accurate probabilities, would require a major investment of effort before these goals could be met.

More practical would be something more closely approximating current practice. This means qualitative measures rather than numerical probabilities, and probably also means only providing these measures for a relatively small number of discrete values of A and e, and for only a few broad categories of violation.

4.2.2. Factors Affecting an Assurance Rating

Assurance is created by adhering to reliable procedures, proven design methods, thorough testing and analysis, and detailed planning and documentation. The CTCPEC identifies twelve subheadings under three categories of assurance of trusted computer systems. The synopsis presented here is derived from the requirements for the highest level defined in the criteria, but should not be taken as complete or authoritative. The listed points relate to the highest assurance level defined in the criteria; lower levels omit or weaken these, down to level 0 which has no requirements at all (and therefore represents no assurance).

A. Operational Trust

1. Product Architecture
   a. The system must be constructed such that the Trusted Computing Base (that portion responsible for enforcing security) is isolated and protected.
   b. The TCB shall be structured into well-defined modules, and layering, abstraction, and data hiding shall be used in its construction
   c. The modules shall operate under a least-privilege principle.
   d. Distinct address spaces shall be used to separate processes and objects.
   e. User interface shall be fully defined.
   f. Protection mechanism shall be simple and complete, with precisely defined semantics.

2. Covert Channel Analysis
   a. Thorough search for covert channels using formal methods.
   b. Provide a measurement or engineering estimate of the bandwidth of each identified channel.

3. Trusted Facility Management
   a. TCB shall support separate operator and sysadmin roles.
   b. Sysadmin functions shall be identified.
   c. No non-essential non-sysadmin functions may be performed in sysadmin role.

4. Trusted Recovery
   a. Shall be assured by recovery procedures and mechanisms.

B. Life Cycle Trust

1. Security Testing
   a. Design shall be analyzed by knowledgeable reviewers.
   b. TCB implementation shall be demonstrated to be consistent

with FTLS.
   c. No design flaws and only few correctable implementation flaws permitted.
   d. Product security features shall work as described in documentation.

2. Design Specification and Verification
   a. Security policy formal model shall be created and maintained over the lifetime of the product.
   b. DTLS and FTLS for TCB shall be provided and shown to be consistent with the model.
   c. Verification shall be consistent with CSSC-endorsed specification and verification system.

3. Configuration Management
   a. Configuration management system shall be used to control all changes to the formal model, DTLS, FTLS, other design data, documentation, source code, object code, and testing materials.
   b. Master copies shall be protected from tampering by a combination of technical, physical, and procedural safeguards.

4. Trusted Distribution
   a. A facility shall be provided to maintain the integrity of distributed TCB code with respect to the master copy.
   b. On-site procedures shall be provided to assure that distributed updates exactly correspond to their master copies.

C. Documentation

1. Security Features User's Guide
   a. User documentation of TCB protection mechanisms, their use and interactions, shall be provided and collocated.

2. Trusted Facility Manual
   a. A secure operations manual, describing facilities and roles, and giving guidelines for secure operation, shall be provided the EDP product administrator.
   b. The manual shall give detailed audit record structure and identify the TCB modules containing the reference validation mechanism.
   c. The manual shall give procedures for secure initial startup, for secure recovery from failure, and for generation of a new TCB after installing modifications.

3. Test Documentation
   a. Developer shall provide evaluators documentation of the test

plan, procedures, and results.

b. Documentation shall include results of the FTLS-TCB code mapping.

4. Design Documentation

   a. Documentation of protection philosophy and its translation into the TCB shall be provided.
   b. TCB module interfaces shall be described.
   c. Formal security model shall be given and proven to enforce the security policy.
   d. TCB mechanisms shall be demonstrated to satisfy the model.
   e. Documentation shall indicate how TCB enforces the reference monitor concept, how it enforces least privilege, and how it is tamper resistant, non-bypassable, and correctly implemented.
   f. Covert channel analysis results and corrective actions shall be documented.
   g. Auditable events used in exploiting covert storage channels shall be identified.
   h. Mechanisms strictly internal to the TCB and not visible in the FTLS shall be clearly described.

The notion of assurance contained in these criteria is defined with respect to a policy as the goal. It does not concern itself with the suitability of the policy in relation to the ultimate objectives, and so does not encompass the upper layer of the design hierarchy of figure 4.2, duplicated here for the convenience of the reader. It is also specific to security policies and to electronic data processing equipment, and many of the points it makes are particular to that design area.

It is possible to look beyond some of this specificity, and to distill some general assurance principles from this list, by classifying the points according to the following scheme. Some of them could justifiably be moved to other classifications, and indeed it could be argued that some deserve to be listed under more than one heading, but for our immediate purposes we will consider the following provisional classifications:

Functional

Includes:
A1a, A1c, A1d

These are specific functional requirements affecting assurance, and come in two forms. One is the prescription of particular user-oriented functionality in the target system. Functionality is, of course, fundamentally a guarantee-related rather than assurance-related issue, but the restriction of implementation choice can be viewed as an assurance-increasing activity in appropriate circumstances.

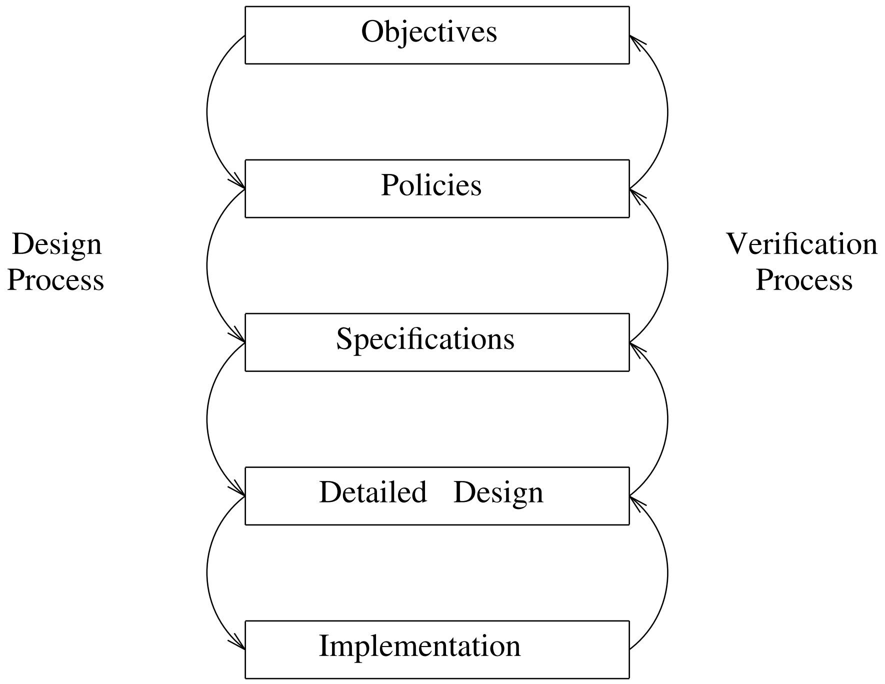


Figure 4.2: Refinements in System Design.

This can occur in an area where the policy is relatively fixed, and where the authority believes itself to be adequately familiar with the range of practical options available, that it feels able to assert that one implementation choice is superior to the others. Adopting that choice then amounts to using a familiar design, with (presumably) a good track record, and for which the authority already has confidence of its reliability and effectiveness.

It may also be the case that the authority is simply unsure of its ability to evaluate other approaches, and wishes to restrict the systems considered to ones for which evaluation techniques and expertise are available. This points out an important issue in ‘‘real-world’’ assurance, namely that it is always defined from the viewpoint of some particular responsible authority that needs to be convinced.

A second form of assurance-related functional requirement is one that is not directly oriented toward the users and their immediate needs, but is employed internally by the system to increase its assurance. Usually, though not necessarily, these involve maintaining the integrity of critical parts of the product, and the provision of increased reliability and tamper resistance.

Familiarity and track record arguments are again valid here, not just regarding the choice of functionality, but also with respect to the expectation that the beneficial effects will be observed.

Architectural

Includes:
A1b, A1f, A3a, A3c, A4a
B3b, B4a, B4b

These are looser restrictions on the type of design that is acceptable to the authority. They may require that certain problems be addressed, or that a particular internal structure be used in the design, but do not specify particular solutions or functionality.

Architectural restrictions affect assurance in a way similar to the first type of the functional restrictions, but in this case the authority is willing to tolerate more variability among the implementations. Possible reasons for this increased tolerance are:

i) there is no clearly superior solution among the candidates once the loose restriction is in place,

ii) the correct choice depends on particulars of the application or of other design choices that are not dictated by the authority,

iii) the authority is confident about its ability to evaluate a range of implementations rather than only a single one,

iv) a combination of the above.

Methodological

Includes:
A1e, A2a
B1a, B1b, B2a, B2b, B2c, B3a
C4c, C4d

These restrict the types of design tools or methods, or verification or testing procedures that may be used, rather than the designs themselves. In some cases, these concern the use of assurance-increasing state-of-the-art techniques (eg. A2a requirement for formal methods), in others is is more a question of organization and good engineering practice (eg. A1e fully defined interfaces).

Documentary

Includes:
A2b, A3b
B1d
C1a, C2a, C2b, C2c, C3a, C3b, C4a, C4b, C4e, C4f, C4g, C4h

Here we have a very different type of assurance, one that emphasizes the fact of communication over the particular content. Some instances of it, like the test and verification documentation, are really just requirements for specific evidence of compliance with Functional, Architectural, or Methodological assurance requirements, and might actually best be considered as part of those rather than as a separate class of requirements.

Other, such as the requirements for manuals giving procedures for secure startup, operation, and recovery, are the primary components of the operational assurance that was earlier identified as belonging to the Detailed Design–Realization guarantee level. Clearly, the issue there is not simply to create appropriate procedures, but to ensure that they are followed, for which communication to the users and administrators is vital.

There are also one other type of document assurance requirement, one that is typified by the requirements for documentation of data structures, internal interfaces, and the protection mechanisms and philosophy. These do not restrict the design or method of design, but rather require disclosure of whatever is designed. There is the suggestion here that it is less important exactly what is produced, than it is that the authority or the users be fully informed.

It is difficult to rationalize this latter type of documentary assurance with the definition that we have been trying to apply, that assurance is related to the reliability of a guarantee. This is particularly true when the disclosure is required to be made to the authority, rather than to the user (the latter situation may be ascribed to operational assurance).

In fact, we believe that this form of documentary requirement is not in place to increase the assurance of the guarantee, but to allow the authority to reliably determine that assurance. In other words, the provision of these documents increases the assurance of the assurance rating. This is unquestionably a desirable goal, but one that is not contained in our working definition of assurance.

Historical

Includes:
B1c

This single point was initially classified as a Methodological restriction, but is subtly different from the rest of those. The idea here, which is admittedly not well expressed by the label ‘‘Historical’’, is that past performance can be used as a predictor of future events. Thus, it is considered better that analysis reveal no design faults than if it does reveal some that are subsequently corrected. The existence of faults calls into question the reliability of the design process, and a faulty design process could have generated more as yet undiscovered errors, is the theory that underlies this determination.

Credentials

Absent for the most part from the CTCPEC assurance list, but present in other similar and related documents, are requirements that specify the training, trustworthiness, or other characteristics of the individuals responsible for the design, construction, testing, and distribution of a system. The only CTCPEC requirement that touches on this is B1a, which requires expert review of the design.

We have classified B1a as ‘‘Methodological’’, partly because it specifies a method (critical design review), but the call for knowledgeable reviewers is also related to the method rather than the result, and so might be independently classed as Methodological as well. On the other hand, it might validly be argued that it deserves to be called by a different name, perhaps ‘‘Meta-Methodological’’, by the following logic: the authority may specify functionality, but failing that may specify design architecture, but failing that may specify design method, and finally in areas where even design methods are left unspecified the last guarantor of quality and reliability is the competence of the individuals who are entrusted to make those choices.

Similar in some ways are requirements for satisfactory background investigations of staff. There the concern is with malign intent rather than incompetence, but excepting that difference the two are quite alike.

It is also possible to analyze these factors along other axes than our set of Functionality, Architecture, Methodology, Documentation, History, and Credentials. We could also consider the design levels between which assurance is affected by each point, and for the most part that classification would be orthogonal to the previous one. Alternatively, we could use the correctness assurance classifications that the ITSEC employs:

- Development Process
    - Requirements
    - Architectural Design
    - Detailed Design
    - Implementation
- Development Environment
    - Configuration Control
    - Developers Security

Operation

- User documentation
- Administration documentation
- Delivery and configuration
- Start-up and operation

Because the ITSEC does not define a narrow range of national security style security policies as do the TCSEC and CTCPEC, it does not contain Functional or Architectural types of elements. The Requirements classification contains constraints which, in terms of the six-axis and design levels classifications, are mostly Methodological constraints at the Policy–Specification boundary. The Developers Security classification contains restrictions that are an intersection of Credentials and Methodology, in that they do not specify particular procedures for security approval of personnel, but they do require the developer to have such procedures in place and to justify their efficacy.

The ITSEC also differs from the others in that it contains a set of criteria relating to the effectiveness aspect of assurance. In particular, the **Suitability of Functionality** factor involves ‘‘determining whether the security functions and mechanisms will in fact counter the identified threats to the security of the target of evaluation’’, which is an Objective–Policy issue.

Demonstration of the internal consistency of a level of specification is also an assurance generating operation, although not identified as such by the CTCPEC. The ITSEC, in its ‘‘Requirements’’ section, demands internal consistency of a formal security target definition, and separately in its **Binding of Functionality** section, requires a demonstration that the set of selected security functions are nonconflicting.

Since these operate within a single specification level, they do not directly correspond to assurance as we have been defining it. They do so only obliquely, in that an inconsistent implementation cannot make any assured guarantees with respect to any (presumably self-consistent) specification. If the specification itself is internally inconsistent, then no guarantee that refers to it can have any assurance, because no implementation can simultaneously satisfy contradictory goals. Whether or not this situation represents a true deficiency in the definition of assurance, is unclear. The current definition is not incorrect, but it might be improved by a restatement that explicitly refers to internal consistency.

If we expand our range of view beyond computer security criteria documents, there are many sources of authoritative quality assurance guidelines and standards, some of which are or may be mandated for suppliers to certain agencies or customers. These, too, might be analyzable in the terms described above. A number of these documents are listed in last section of this chapter.

4.2.3. Forming an Assurance Measure

Having identified a large number of assurance-generating activities, the task remains of marshalling them into a coherent overall assurance measure for a developed system. This is complicated by the fact that the factors are interdependent, some being associated with design, others with production, distribution, startup, recovery, operation, and ongoing maintenance, and many of these may be necessary to ensure the initial, continual, and effective assurance of any guarantee.

At any level in the design hierarchy, it is necessary to concern ourselves about initial, continual, and documentary aspects of assurance. At the lowest levels (the Detailed Design–Implementation interface), we must also consider operational assurance. The reliability of a guarantee is only as good as the weakest of these, and so they must all be present in adequate measure in order to generate any substantial level of assurance.

First, initial assurance must be supplied. This can come from functional constraints, or by means of architectural or methodological constraints that directly affect the processes of design, review, verification, or testing of the design. Initial assurance can also provided by historical arguments applied to the results of testing or verification exercises. Also, the credentials of the designers, reviewers, testers, and verifiers can be considered as further evidence of reliability of the initial design.

Secondly, it must be demonstrated that the validity of the guarantee is ongoing. Contributions here may be made by functional, architectural, or methodological constraints affecting the design environment, and impacting on the integrity of the specification, working documents, or resultant implementation.

At the higher levels of the design hierarchy, at issue is the integrity of the documents describing the design rationale and results. These must be safeguarded, not only from tampering, but also from irregular modification by otherwise authorized personnel, eg. the application of "bug" fixes or extensions that are not generated through an approved assurance maintenance procedure.

At the lower levels, in addition to the above, the results of the design or fabrication are replicated, distributed, maintained, and used. Since each of these activities provides a new opportunity for the deterioration of assurance, they must each be adequately assured. This involves quality control procedures during replication, on-site integrity checking of the distributed copies, similar precautions for the generation, distribution, and application of updates, as well as operational integrity protection whether supplied by the system itself or by its operating environment.

Documentary assurance can be appropriately dealt with if the documentation is viewed as a companion product, one which is designed along with the primary product but which has its own proper initial integrity, maintenance, duplication, and distribution problems. Part of the documentation, that part consisting of descriptions of the design, philosophy, methods, and analysis, testing, and verification results, directly supports the determination of the primary product initial assurance. At the Detailed Design–Implementation level, user documentation also enters the picture in support of the continual (operational) assurance of the primary product.

Given a guarantee G and a hypothesized faulty guarantee G' (that is, one in which the implementation actually satisfies a different policy than that which it is intended to satisfy), a set of initial assurance-providing procedures A_I, and a postulated enemy effort e, assume that we can derive a violation probability of the ideal form:

$$Prob_{initial\ \ violation}\ (G, G', e \mid A_I)$$

This was rewritten as a probability conditional on the A_I term, for reasons which will become obvious later. Note that this rewriting is only a cosmetic modification of the expression, since it has always been the case that the probability be calculated assuming adherence to the assurance-generating procedures represented by A_I.

Also of interest is the level of the continual assurance of the product. Let the assurance-generating procedures here be represented by A_C. It refers to an implied guarantee

> *The assurance of the guarantee G does not degrade over time.*

Let us call this implied guarantee C_G, and for the sake of simplicity consider only the simplest, non-specific failure mode, where the assurance of G does fail (in some undefined way) over time. Then, based on the assumed use of procedures A_C to provide continual assurance, we also have the probability

$$Prob_{subsequent\ \ loss\ of\ assurance}\ (C_G \mid A_C)$$

Documentary evidence supporting the assertion of A_I is itself the object of assurance-generating procedures, but in this case the assurance is associated with the implied auxiliary guarantee $VALID_{A_I}$:

> *A_I accurately represents the procedures used in implementing guarantee G.*

The reliability of this guarantee can be of a much simpler form than the others since the documents DOC_I are themselves evidence, rather than depending in turn on some other implied guarantee. Therefore, given the existence of a particular set of such documentation, assume the assignment of a failure probability

$$Prob_{unsupported}\ (VALID_{A_I} \mid DOC_I)$$

which indicates the likelihood that, in spite of the specified evidence being supplied, the procedures described by A_I were in fact either not followed or not correctly followed.

Similarly, other documentary evidence supports the assertion of A_C. This evidence consists mainly of user and administrator manuals indicating the proper ways to use the system in order to maintain a high assurance of proper functioning. In this case, the corresponding failure probability is

$$Prob_{unsupported}\ (VALID_{A_C} \mid DOC_C)$$

by similarity to the previous case.

These four quantities combine in the following way, if we can assume that the issues relating to initial versus continued assurance have been adequately separated, such that failures of G and of C_G are statistically independent. The total probability that the initial guarantee G may fail according to G', based on the available evidence DOC_I, is

$$\begin{aligned} Prob_{initial\ \ violation}\ (G, Gprime\ , e \mid DOC_I) = \\ & Prob_{initial\ \ violation}\ (G, G', e \mid A_I) \\ & \times\ (1 - Prob_{unsupported}\ (VALID_{A_I} \mid DOC_I)) \end{aligned}$$

Also derivable is the probability of some subsequent (undifferentiated) guarantee compromise, based on the observed documentation. This takes the form

$$\begin{aligned} Prob_{assurance\ \ loss}(C_G \mid DOC_C) = \\ & Prob_{subsequent\ \ loss\ of\ assurance}\ (C_G \mid A_C) \\ & \times\ (1 - Prob_{unsupported}\ (VALID_{A_C} \mid DOC_C)) \end{aligned}$$

4.3. COMPOSING ASSURANCE

We have previously suggested two ways in which assurance measures may be composed. One is vertical, creating a composite assurance over all the levels in the design hierarchy corresponding to the composite guarantee linking the physical realization of a system to its ultimate policy objectives. The second is structural, to determine an assurance measure for the overall guarantee of a system from the overall guarantees and assurances of its components.

4.3.1. Vertical Assurance Composition

First is the vertical composition to establish the ''chain of belief'' from a lower-level specification all the way back to the original Level 1 objective. This composition generates the cumulative assurance value that was required for the measure of guarantee quality described elsewhere in this report.

Since the initial and continuing aspects of assurance are independent contributors to total assurance, and since the division between them is consistent through all levels of the design hierarchy, they may be separately composed to form cumulative initial assurance and continuing assurance measures. This remains a valid distinction even though the design and verification processes occurred at the different levels at very different times. It does, however, mean that an cumulative initial assurance value does not correspond to an actual assurance at any particular instant. Only the combination of initial and continuing assurance has a meaningful interpretation when dealing with the cumulative assurances.

With initial assurance, the vertical composition problem is difficult, but this difficulty does not result from any inherent property of initial assurance. Rather, it results from our earlier decision to specify particular guarantee failure scenarios, so that we would be able to distinguish between different levels of severity of failure.

Let us assume a sequence of specifications S_1 through S_5, where S_1 is the Policy Objective level statement, and S_5 is the corresponding Implementation. Guarantees exist between each pair of adjacent specifications. According to the numbering scheme we have been using, the guarantee of correspondence between S_i and S_{i+1} is called the *level i guarantee* , and consists exactly of the statement

$$G_i = \text{``}S_{i+1} \textit{ satisfies } S_i\text{''}$$

Associated with each of these is an initial assurance measure, expressed as a function depending on the conjectured deviation of that level's guarantee. The chain of guarantees combines straightforwardly into the cumulative guarantee

$$G = \text{``}S_5 \textit{ satisfies } S_1\text{''}$$

and, corresponding to this, we would like to combine the four initial assurances

$$Prob_{initial\ \ violation}\,(G_i, G'_i, e \mid DOC_{I_i})$$

for i from 1 to 4, forming an overall assurance measure

$$Prob_{initial\ \ violation}\,(G, G', e \mid DOC_I)$$

The difficulty that now arises is that of relating deviations in the overall guarantee to the deviations along the ''chain'' that may give rise to them.

Measures of continuing assurance combine more easily than this, because when defining them we made the simplifying assumption that different failure modes are not distinguished. Then, since validity of the chain of assurance requires the continuing validity of each link, and assuming independence of the possible failures, we have

$$CumProb_{\text{loss of assurance}} = 1 - \prod_{i=1}^{N} \left(1 - Prob_{\text{assurance loss}}(C_{G_i} \mid DOC_{C_i})\right)$$

where the C_{G_i} and DOC_{C_i} are, respectively, the implicit guarantee of continuing validity at level i, and the submitted documentation supporting that implicit guarantee at level i.

Several possibilities are diagrammed in figure 4.3. In each, the vertical direction represents different specification levels, from Policy Objective at the top (level 1) to Implementation at the bottom (level 5). The horizontal axis is an attempt to represent ‘‘specification space’’, such that one point beside but separated from another represents a distinct specification at the same level. The heavy points are the specifications at five levels, and the arrows represent the inter-level guarantees. These arrows are drawn upward

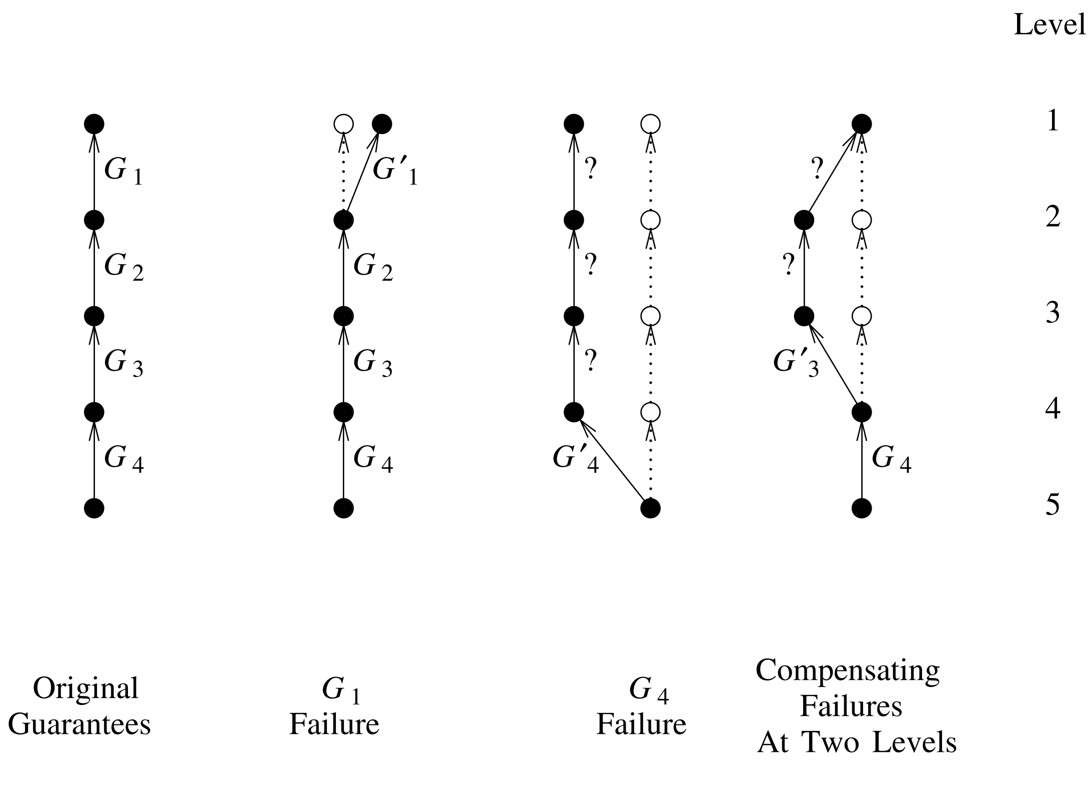


Figure 4.3: Perturbations of Vertically Composed Guarantees.

because the guarantees represent linkages from a lower-level specification to its immediate superior.

The leftmost example is the base case, where the overall guarantee G is a combination of the unperturbed guarantees at levels 1, 2, 3, and 4. For this case, we can determine the appropriate probability,

$$CumProb_{initial\ violation}(G, G, e \mid DOC_I)$$
$$= 1 - \prod_{i=1}^{N}(1 - Prob_{initial\ violation}(G_i, G_i, e \mid DOC_{I_i}))$$

using the unperturbed assurance measures (i.e., those in which $G'_i = G_i$) for each component guarantee.

In the second example, we have postulated a faulty level 1 guarantee component which, instead of establishing the desired Policy Objective, actually satisfies a slightly different one. Its perturbed guarantee, G'_1, reflects this by being ‘‘deflected’’ off target in the specification space. This perturbation is one that was provided for in the level 1 assurance determination. The probabilities of failure contemplated there were predicated on guarantee failures of exactly this form, with the lower-level specification implementing a higher-level policy that differs from the one desired. We can therefore use the level 1 initial assurance results to calculate the probability of initial violation of G_1 failing as G'_1, then combine it with the no-violation probabilities for the rest of the levels, and compute an overall assurance value for

$$CumProb_{initial\ violation}(G, G', e \mid DOC_I)$$

However, the third case shown does not share this good behaviour. It contains a guarantee violation at the lowest level, between the Detailed Design and Implementation levels. Its implementation, instead of being a faithful refinement of the correct target specification at the Detailed Design level, corresponds to a different Detailed Design. Now, although that particular link in the guarantee chain does fit within the range of contemplated failures, the links above it do not.

Remember that, when defining assurance measures for individual levels, it was assumed that the ‘‘bottom end’’ of the guarantee arrow remained fixed. Thus, the level 3 assurance measures that are presumed available in this case all assume that the level 4 specification is the designed-for specification. But, since the level 4 guarantee has been perturbed, the level 3 guarantee arrow no longer has that specification as its origin. Hence, we cannot determine the probability of this particular cumulative guarantee by combining those assurance measures evaluated for appropriate perturbations.

It appears that the most that can be accomplished, by combining in any straightforward way the available per-level assurance measures, is to do a binary calculation of the probability of success versus that of failure, without attempting to quantify the degree of failure. We can calculate the probability of success if we assume it is equal to the probability of the first case in Figure

4.3; this is not precisely the correct value, since it does not account for the case, shown in the fourth diagram of the figure, where compensating perturbations at two levels cancel one another. Neglecting this case amounts to assuming that the frequency of such fortuitous multiple errors is negligible.

4.3.2. Structural Assurance Composition

The second form of assurance composition that must be addressed is the derivation of a suitable assurance measure for the guarantee of a system composed as discussed elsewhere in this report. We will assume in this section that the composite guarantee was determined by restricting the scope of the problem to a suitably well-behaved range of properties, components, and compositions.

If the composition is of the simplest sort, in which the properties of the composite are simply some collection of properties of the components, then it is possible that the assurance will compose in a similarly simple manner. For this to occur, it is necessary that the perturbations of the component guarantees be benign enough that the components remain functionally independent.

Examples of how this can fail to happen abound, as for instance in a mechanical system where one component gets sufficiently out of alignment that it begins to interfere with the motion of another, or in a software system where a wild array reference corrupts data belonging to another module.

It cannot be expected, in general, that the magnitude of a guarantee perturbation can be used as a reliable indicator of the likelihood of interference of this kind. The guarantee relates to the overall functionality of a component, while the interference depends on the specific modes of failure which, considered in terms of their effect on guarantee, may not be distinguishable from other less virulent failures. The mechanical component, if differently misaligned, might display a similar functional discrepancy but no longer interfere with its neighbour. The wild array reference may have a catastrophic, mild, or barely noticeable effect on the operation of its own module, arbitrarily combined with a similar range of effects on its neighbour.

Rather than trying to predict these effects, a more sensible approach is to construct the components, or to compose them, such that the risk of interference is lessened. This may be accomplished by, in some sense, increasing the ‘‘distance’’ between components, or equivalently by isolating them with barriers that prevent unplanned interactions. These efforts are reflected in the specification if it is complete, and hence in the guarantee, since a full specification indicates the environmental assumptions under which it is valid, and the provision of barriers renders the component insensitive to most environmental effects.

Once interference has been effectively ruled out, the exact form in which the assurance elements are composed depends on the statistical independence of the individual guarantee perturbations. This, however, is not necessarily evident from their assurances, since it depends on the mechanisms responsible for the perturbations. Consider a system built out of two identical components. Failures due to systemic error, such as design faults, will track exactly on the two subsystems, so that the probability of initial violation in the composite due to a cause of this kind will equal the probabilities of the same violation for a single component.

However, it may well be the case that a functionally indistinguishable fault may occur as a result of hardware failure, which may be considered to occur independently in the two components. In that case, the probability of partial system failure is equal to that of single component failure, but the probability of total system failure is much less.

The next simplest situation, after independent components, is that in which there are preplanned interdependencies among the components. That is, cases where the components are given conditional guarantees that anticipate the function of the components they are to be composed with. In these situations, the interference problems are augmented by uncertainties about the sensitivity of the dependent components.

A minor irregularity in the output of one component may cause a second dependent one to fail catastrophically, depending on how sensitive the latter is to variations in the input it receives from the former. A data channel that occasionally flips a bit, if the data it carries is sufficiently critical, may render the entire system unusable. Or, if an interdependency loop exists in the system, positive feedback may escalate a harmless deviation into a total system failure.

One approach is, again, to design for the contingency as was suggested for dealing with interference. Here, it means equipping the critical user of the bit-flipping data channel with integrity checking functions, so as to minimize its sensitivity to the channel's errors.

Sensitivity analysis is another possible means of addressing this problem. An estimating form of this is fault simulation, similar to that employed in IC design, in which specific component defects are assumed and the effect on overall system performance is derived. A number of such simulations could give some confidence in an estimate of the fault sensitivity of the system. Effectively, this approach involves applying a new set of assurance generating procedures to the composed system, in addition to those already applied to the components, so is not simply a matter of combining available component assurances. It also requires the *significant manipulation of the specifications* that we wished to rule out. On either of these grounds, it is not the kind of approach that this work is striving to develop, but nevertheless a variation of it may be the most effective means of establishing assurance in the composed system.

Perhaps a more acceptable, though not fundamentally different, method would involve redefining the component assurance measures so that they included an input sensitivity function, that would map differences in the environmental assumptions into corresponding guarantee perturbations. In the absence of feedback, these sensitivities could then be used in a fairly straightforward fashion to combine the component assurances. This is described as ‘‘not fundamentally different’’ because it still involves the same manipulation of the specifications as before, but now this manipulation is performed once when the component assurance measure is being generated rather than at each composition.

Finally, the case of full emergent properties is not addressed. Since it is not possible to even generate the composite guarantee in this case, at least not without opening up the detailed component specifications, it is felt that it would be unnecessary to attend to the question of assurance composition under these circumstances.

4.4. QUALITY ASSURANCE PROGRAM STANDARDS

The following documents are quality assurance standards from a variety of sources world-wide, and with a wide variety of scopes.

Some are applicable to very narrowly defined subjects, or to limited areas within design, manufacturing, operations, or maintenance. The more detailed ones may even go to the level of specifying the schedules and procedures for the calibration of test equipment. Others are much more general and encompassing, and impose requirements on such things as the upper management structure of a compliant organization.

All are representative of some authority’s view of which activities and pieces of evidence contribute to quality assurance. A comprehensive theory of assurance should be able to deal with the issues they raise as well as those contained in or implied by national criteria for trusted product evaluation.

Canadian Standards Association (CSA)

CAN3-N286.0 — Quality Assurance Program Requirements for Nuclear Power Plants.

The following four entries represent a set of successively less stringent quality assurance requirements. They are accompanied by their descriptions from the companion overview document.

CAN3-Z299.1 — Quality Assurance Program — Category 1

The Standard aims at preventing the **occurrence** *of nonconforming products or services. This is achieved by thorough planning and controls which extend to identifying and correcting weaknesses in the Program. The Standard is suitable for custom-designed, high technology products and services. They tend to require many complex processes* *and extensive design effort by either customers or suppliers, or both. Failure in service could result in extreme cost or undue risk to health and safety, or both.*

CAN3-Z299.2 Quality Assurance Program — Category 2

This standard aims at reacting to nonconforming products or services to prevent their **recurrence** *. This is achieved by specifying feedback control to correct causes of nonconformances. This Standard is suitable for relatively high technology products or services. They tend to require design verification and production planning, and have a significant number of complex processes. Failure in service could result in serious cost or significant risk to health and safety, or both.*

CAN3-Z299.3 Quality Assurance Program — Category 3

This Standard requires suppliers to plan and establish a program for verifying the conformance of products or services throughout the process. The program is documented but in a limited manner. This Standard is suitable for products or services requiring some complex processes. They may be high volume services or mass produced products. Failure in service could result in significant cost or some risk to health and safety, or both.

CAN3-Z299.4 Quality Assurance Program — Category 4

This Standard requires suppliers to plan and establish a program for sorting the good from the bad. The program need not be documented unless specified in the contract. This Standard is suitable for mass produced products or for high volume services designed to commercial technical standards or for simple processes such as custom machining, assembly, and installation

Canadian Department of National Defence (DND)

D-02-001-002/SF-001 Quality Program Requirements for Contractors — DND 1015

DND 1016 Contractor's Inspection System Requirements

DND 1017 Basic Inspection Requirements for Contractors

Note: DND now specifies NATO AQAP Standards.

NATO Allied Quality Assurance Publications

AQAP-1 Quality Control System Requirements for Industry

AQAP-4	Inspection System Requirements for Industry
AQAP-9	Basic Inspection Requirements for Industry

International Organization for Standardization (ISO)

ISO 6215	Nuclear Power Plants — Quality Assurance
ISO 9001	Quality Systems — Model for Quality Assurance in Design/Development, Production, Installation and Servicing.

Corresponds approximately to CAN3-Z299.1

ISO 9002	Quality Systems — Model for Quality Assurance in Production and Installation.

Corresponds approximately to CAN3-Z299.2

ISO 9003	Quality Systems — Model for Quality Assurance in Final Inspection and Test.

Falls somewhere between CAN3-Z299.3 and CAN3-Z299.4

ISO 10011-1	Guidelines for Auditing Quality Systems

International Atomic Energy Agency (IAEA)

50-C-QA	Quality Assurance for Safety in Nuclear Power Plants

United States Department of Defense (DOD)

MIL-Q-9858A	Quality Program Requirements
MIL-I-45208A	Inspection System Requirements

British Standards Institution (BSI)

BS 5750 Part 1	Specification for Design, Manufacturing and Installation
BS 5750 Part 2	Specification for Manufacture and Installation
BS 5750 Part 3	Specification for Final Inspection and Test

American National Standards Institute (ANSI)

NQA-1	Quality Assurance Program Requirements for Nuclear Power Plants

American Society for Quality Control (ASQC)

ASQC C-1	General Requirements for a Quality Program.

United States Nuclear Regulatory Commission (NRC)

10 CFR 50	Quality Assurance Criteria for Nuclear Power Appendix B: Plants and Fuel Reprocessing Plants.

American Society of Mechanical Engineers (ASME)

The following citations are from the Boiler and Pressure Vessel Code

Sect. 1-A300	Power Boilers: Quality Control System
Sect. 3-NCA3800	Nuclear Power Plant Components: Metallic Material Manufacturer's and Material Supplier's Quality System Program
Sect. 3-NCA4000	Nuclear Power Plant Components: Quality Assurance
Sect. 4-App. F	Heating Boilers: Quality Control System
Sect. 8-App. X	Pressure Vessels: Quality Control System

5 SURVEY OF FORMAL MODELS

Turn the pages of your Greek models night and day.
Horace,
Book III (Ars Poetica),
8 BC.

What we experience in nature is in models, and all of nature's models are so beautiful.
Buckminster Fuller,
Profile in The New Yorker,
1966.

5.1. INTRODUCTION

In this chapter, we review the existing formal approaches to security modelling, and in particular the issue of the composability of models.

Widespread interest in security composition was generated by the appearance of a 1987 paper by Darryl McCullough entitled ''Specifications for Multi-Level Security and a Hook-Up Property'', in the *Proceedings of the 1987 IEEE Symposium on Security and Privacy* . He made the surprising observation that subsystems, secure by a popular theoretical definition of confidentiality, and interconnected in a straightforward and intuitively reasonable way, might interact such that the resulting system would not meet the given security condition.

Since then, other authors have continued and extended the investigation of these ''hook-up'' phenomena. Since the body of pre-existing work on this topic is small, it must be studied in detail to yield the insights that will lead to further advances in the area. This chapter contains in-depth reviews and criticisms of the most salient of this existing work. It will be seen to evidence a profound comprehension that will form a firm foundation for future work.

5.2. McCULLOUGH'S COMPOSABILITY

In a paper presented at the 1987 IEEE Symposium on Security and Privacy, McCullough introduces the concept of *hook-up security* as a viable approach to building *composable trusted systems* .

5.2.1. Multi-Level Security and a Hook-Up Property

McCullough proposes the following circular definition:

> A system is hook-up secure if it is deducibility secure and if, when it is hooked up with a second hook-up secure system, the result is a hook-up secure composite system.

This is the desired (security) functionality of a composite system. The definition implies that deducibility security is at least a necessary condition for hook-up security. In another paragraph, McCullough states that, based on attempts at proving the security of systems composed of deducibility secure systems, it is not a sufficient condition.

The remainder of the paper is devoted to finding a requirement on individual systems that guarantees the desired functionality is maintained on hook-up.

Three models of multi-level security are discussed: Bell and LaPadula's access controls [Bell74], Sutherland's deducibility model [Sutherland86], and Goguen and Meseguer's non-interference requirement. [Goguen82]

The major difficulty with using the Bell-LaPadula model as the basis for hook-up security is that it defines a system's *trustworthiness* only in terms of its attention to labels placed on users and data. Systems that must be trusted to handle multi-level data in a general way must be authorized by some mechanism not included in the Bell-LaPadula model, and thus it is not adequate as a basis for the hook-up security requirement.

5.2.2. Deducibility

McCullough describes deducibility in terms of the *traces* of a system, which are the possible sequences of inputs and outputs which characterize the system's operation. To simplify the discussion, a two level security scheme is assumed.

The fundamental notion of deducibility is that a *low-level* user, who can be assumed to know the system's operating principles, and who can view all low-level inputs and outputs, should be prevented from *deducing* anything about the inputs made by *high-level* users.

The set T is the set of all traces τ achievable by the system. These traces are the interleaved sequences of all low-level and high-level input and output events. No absolute timing information is preserved by this system model; only the relative temporal ordering of events. Events themselves may be arbitrarily complex, from binary signals to messages over a network to graphical displays, but are unidirectional input or output transactions.

For each trace $\tau \in T$, a trace τ_{low} can be formed, which is τ purged of all high-level events. The set of all τ_{low} for a given T is L.

Similarly, for each trace $\tau \in T$, a trace τ_{high} can be formed, which is τ purged of all low-level events, and the set of all τ_{high} for a given T is H.

A low-level user can see, during one session with the system, a particular τ_{low}. In the worst case, the low-level user knows T, and can thus form the set T', which is the subset of T which is consistent with τ_{low}. That is, for each member of T', purging all high-level events leaves exactly τ_{low}.

With this information, the low-level user can form the set H', which is the subset of H consistent with T'. That is, for each member of T', purging all low-level events leaves one member of H'.

Deducibility can now be succinctly defined in terms of traces as:

> A system is *deducibility secure* if, for every $\tau_{low} \in L$, the corresponding set H' is equal to H.

Thus, deducibility security means that all possible high-level event sequences must be consistent with every possible low-level event sequence.

A major difficulty with deducibility is that it fails to provide reasonable security in the presence of non-deterministic behaviour. McCullough attempts to demonstrate this with an example of a system which is obviously not secure in an intuitive sense, yet which is claimed to be deducibility secure.

The example system has a high-level user entering text, and a low-level user running a program which, upon receiving a number n from the low-level user, sends to the low-level user either:

- the n^{th} character of the high-level user's text, if it exists, or
- a random character, otherwise.

Intuitively, this system is insecure, since, for example, by entering the numbers $1, 2, 3, \cdots$, the low-level user will receive either the high-level user's text or a random string of characters. Depending on the circumstances, the low-level user may easily be able to differentiate one from the other. Receiving an English sentence, for example, is a good indication that it is not randomly generated. Receiving gibberish, on the other hand, is not conclusive evidence, since the high-level user may be using an enciphering code.

However, the system is not deducibility secure as claimed. If, in response to the number 7, the letter J is received, then, by the system's method of operation, which is known to all users, either the high-level user's text has no 7^{th} letter, or that letter is a J. In the terms of the definition, the set of high-level input sequences consistent with the low-level events is smaller than the set of *all possible* high-level input sequences.

Though the example is flawed, it has been shown elsewhere[Thomson88] that deducibility, as generally understood, cannot be adequately applied to non-deterministic systems.

5.2.3. Non-Interference

McCullough next turns to Goguen and Meseguer's work on non-interference, and again casts the definition in terms of traces.

Fundamentally, non-interference requires that no high-level input can have an effect on low-level events. This is a stronger (more restrictive) requirement than deducibility, which allowed interference, as long as the nature of the high-level inputs was not revealed.

Given a trace τ, modifying it by adding or deleting high-level inputs results in a sequence σ, which is not necessarily a valid trace. It may be possible to construct a valid trace τ' from σ by adding or deleting high-level outputs after its first modified input. Non-interference can be defined in these terms as:

> A system is *non-interference secure* if, for every τ, for every σ which can be formed from it, at least one τ' exists.

This definition, in contrast with that used by previous authors, extends the non-interference concept to non-deterministic systems. The operative phrase in this regard is ''at least one'', which implies that systems may produce multiple distinct output sequences when presented with identical inputs.

An example of a composite system is then presented, which illustrates that hooking up non-interference secure systems may not result in a non-interference secure composite system.

Machine *A* has one high-level input *in*, and one high-level output *out* which is a reply to the *in* following some processing. There is a low-level *cancel* input, which cancels any high-level processing that is underway, and a low-level *ack* output which acknowledges the *cancel* input after some time interval. If there is high-level processing at the time of the *ack*, that is, if there has been no *out* since the last *in*, the high-level processing is terminated, and no *out* will occur until after the next uncancelled *in*. If there is no high-level processing at the time of the *ack*, then a low-level *error* output may be produced at some time following the *ack*; however, the *error* output is not guaranteed to occur.

Machine *B* is similar to *A*, but does not have an *ack* output. It cancels high-level processing, if any, at the moment the *cancel* is received. If there is no high-level processing at the time of the *cancel*, then a low-level *error* output may be produced at some time following the *cancel*; however, the *error* output is not guaranteed to occur.

McCullough does not describe how the machines behave when multiple inputs are made without waiting for a response. Let us state the additional requirement that inputs that have not been fully processed (or cancelled) are ignored. Thus, any *in* is ignored that arrives while high-level processing is occurring. Likewise, any *cancel* input to machine *A* is ignored until the *ack* has been produced. Also, to keep the number of possible traces manageable, if a *cancel* to either machine produces an *error*, any additional *cancel* input before the *error* is ignored.

Then the set of permissible traces for each machine can be enumerated, as has been done in schematic form in figures 5.1 and 5.2. Each trace diagram consists of a time line, running vertically, and events, drawn as labelled arrows along the time line. Time flows *up* the time line; earlier events are nearer the bottom of the time line. Dashed arrows signify high-level events, and solid arrows are low-level events. An arrow directed at the time line is an input event, while an arrow directed away denotes an output event.

Figure 5.1.a shows the *null* trace for machine A, that is, the trace consisting of no events. Figure 5.1.b is the trace consisting of one high-level *in* input followed, some time later, by an *out*. Figure 5.1.c introduces a new symbol $\leftrightarrow$ that denotes ''zero or more occurrences of the trace in figure 5.1.b.'' This trace shows that a *cancel* will eventually be followed by an *ack*, and that if *in* was not the last high-level event before the *ack*, an *error* can be produced. Figure 5.1.h shows that the *error* response is not guaranteed. More traces can be formed by concatenating these traces together.

The traces in figure 5.1.a through figure 5.1.g form the ''deterministic'' traces for machine A. Figure 5.1.h and figure 5.1.i demonstrate that the *error* output may or may not occur, given identical inputs. Similarly, the traces in figure 5.2.a through figure 5.2.d describe a ''deterministic'' version of machine B.

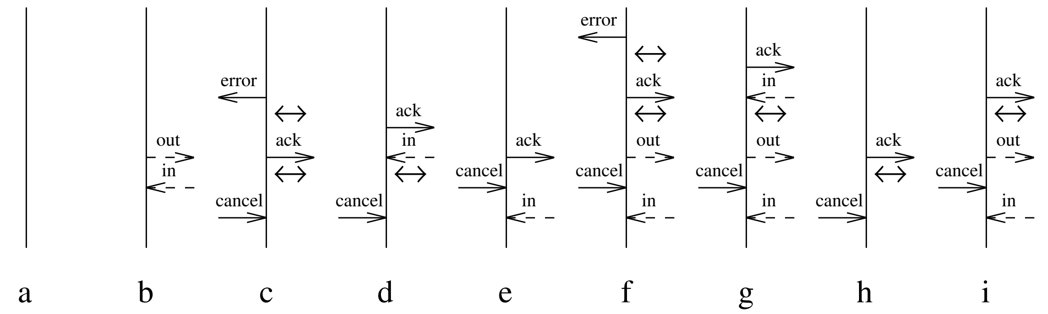


Figure 5.1: Permissible Traces for Machine A.

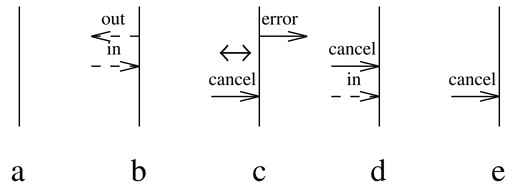


Figure 5.2: Permissible Traces for Machine B.

One can directly apply the definition of non-interference to show that the ''non-deterministic'' traces are necessary for A and B to be non-interference secure. For example, by removing the *in* event from the trace in figure 5.1.e, a sequence results that cannot be converted into a combination of the traces in figure 5.1.a through figure 5.1.g by adding or deleting high-level outputs. However, the sequence is equivalent to the trace in figure 5.1.h, and therefore the definition has been complied with in this case. Showing that A and B are non-interference secure requires demonstrating that every possible high-level input addition to, or deletion from, every trace can be converted into some trace by adding or deleting some high-level outputs. Clearly, applying this definition in practice to any machine more complex than A would require automated assistance.

To hook up two systems, some of the outputs of one become inputs to the other as well. Clearly, high-level outputs can only be fed to high-level inputs, and both systems must agree on the levels of users and data. Outputs fed to inputs become internal events of the composite machine, and effectively disappear from its trace.

McCullough hooks up A and B as follows: A's *ack* output feeds into B's *cancel* input, A's *out* feeds into B's *in*, and B's *out* feeds into A's *in*. A's *in* is also available as an external input, besides receiving B's *out*.

Thus the composite system has one low-level input, the *cancel* input to A, and two low-level outputs, the *error* outputs from A and B. It also has one high-level input, A's *in*, but no high-level outputs.

Figure 5.3 shows several permissible traces of the composite machine. Internal events are shown for reference, but are not properly part of the composite machine's trace.

The trace in figure 5.3.a is one trace (the others are missing one or both *error*s) achievable in the case where no *in* precedes the internal low-level event. The other traces in figure 5.3 are two of the many possible traces in

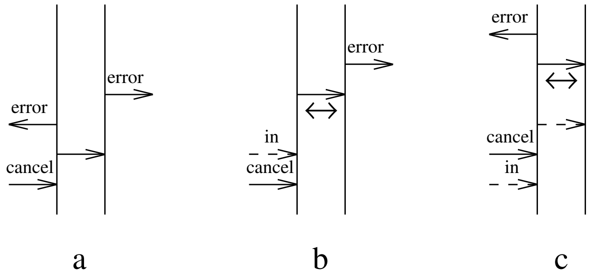


Figure 5.3: Some Traces of Composite Machine.

which the *in* does occur.

Though all possible traces of the composite machine are not enumerated here, it is straightforward to determine that, if *in* occurs before the internal low-level event, the two *error* outputs cannot both be produced. Thus, to demonstrate that the composite machine is not non-interference secure, an *in* may be added to the bottom of the trace in figure 5.3.a. Clearly, this sequence is not a permissible trace, and cannot be converted into one by adding or deleting high-level outputs, since the composite machine has none! According to the definition, then, the composite machine, although composed of non-interference secure components, and hooked up in a reasonable fashion, is not itself non-interference secure.

5.2.4. Hook-Up Security

McCullough attempts to remedy the non-composability of the non-interference notion by making a slight alteration to the definition. This modification is intended to "slow down" the movement of high-level messages, whose bouncing from machine to machine in the last example seemed to be problematic. The definition, in terms very similar to those for non-interference, is as follows.

Given a trace τ, modifying it by adding or deleting high-level inputs results in a sequence σ, which is not necessarily a valid trace. It may be possible to construct a valid trace τ'' from σ by adding or deleting high-level outputs after the last input in, or immediately following, the modified input sequence. Hook-up security can be defined in these terms as:

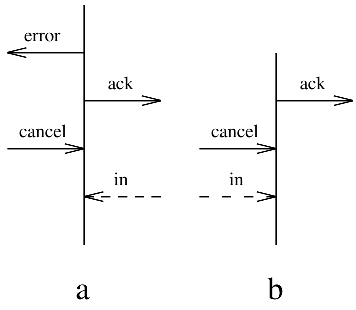


Figure 5.4: Non-Hook-Up Secure Traces.

> A system is *hook-up secure* if, for every τ, for every σ which can be formed from it, at least one τ'' exists.

This is non-interference with the added stipulation that high-level outputs cannot be fixed up arbitrarily soon after a modified high-level input. High-level output ''fixing'' must wait until after any immediately following inputs.

Clearly, since the composite machine from the last example was not non-interference secure, it cannot be hook-up secure. This implies that at least one of the components themselves was not hook-up secure, which can be verified by referring to figure 5.4.

Figure 5.4.a shows a trace of machine A to which has been added a high-level input. There is only one way to convert this into a valid trace for A, and that is to add a high-level output between the *cancel* input and the *ack* output. This fix obeys the hook-up security criterion, and in fact machine A is hook-up secure.

Similarly, figure 5.4.b illustrates a trace of machine B to which has been added a high-level input. There is only one way to convert this into a valid trace for B, and that is to add a high-level output before the *cancel* input. However, this fix violates the hook-up security criterion, and thus machine B is not hook-up secure.

The following theorem wraps up the paper:

> If a system is hook-up secure, then it is deducibility secure, and it has the non-interference property. If systems A and B are both hook-up secure, then any composite system formed by identifying outputs of A with inputs of B with the same securi-

ty level, and vice-versa, is hook-up secure.

This satisfies the circular definition stated in the introduction, and the proof appears in the 1988 paper ‘‘The Theory of Security’’ [McCullogh88a]. This is a lengthy and difficult proof which we will not discuss here. However, we later present a proof from Johnson and Thayer’s ‘‘Security and the Composition of Machines’’ [Johnson88] which is based on McCullough’s, and which proves a very similar property.

5.3. NONINTERFERENCE AND COMPOSABILITY

McCullough’s paper from the 1988 IEEE Symposium on Security and Privacy[McCullough88c] is mostly a reiteration of the concepts he presented a year earlier. It does contain, however, an interesting discussion of the origins and consequences of non-determinism, as well as a new formulation of hook-up security.

5.3.1. Non-Determinism

The essential meaning of non-determinism is that the input sequence to a non-deterministic machine does not uniquely determine the output sequence. This is a problem because it renders otherwise satisfying security formalisms, such as Goguen and Meseguer’s original non-interference model, unusable.

Even when a machine is said to be deterministic, this is often not the case, due to the subtle effects of certain implementation-related mechanisms.

One type of problem arises from the fact that time is not explicitly considered in the trace model. Running a program on a faster machine may produce different results.

Similarly, the details of processor scheduling are usually abstracted away in the course of security modelling, resulting in behaviour that appears non-deterministic to the user. For example, an interactive program may be rendered unusable on a heavily loaded machine.

A related sort of difficulty arises when systems are composed in certain ways. If multiple processes have their outputs merged onto a single channel, the resulting composite output stream will depend on the relative speeds of the processes. Thus the output sequence is not wholly determined by the input sequence to the composite machine.

Another problem resulting from the composition of systems is due to the requirement of buffering between processes. Since all input buffers must be finite in capacity, some policy must be followed when an input arrives for a full buffer. Two alternatives are available: either the new input (or the oldest unprocessed input, or some random input in the buffer) is ignored, or the sending process is blocked until the buffer is no longer full. Both alternatives, unfortunately, lead to difficulties.

If the new input is *dropped*, the output sequence from the intended receiver will be dependent on the size of the buffer and the relative speeds of the processes. Buffer sizes could be modelled (though such ''low-level'' details are usually ignored), but the trace model cannot adequately deal with process speeds, especially since these are highly dependent on the detailed behaviour of hardware devices.

If the sending process is blocked, a potential covert channel has been introduced: a low-level sender can receive information by sending to a high-level process. This information may seem harmless, since an unblocked process does not receive notification that it was blocked, and a blocked process cannot do anything. However, McCullough presents an example of a composite system with two sending processes that gives rise to a very effective covert channel. What he does not mention is that a single sending process can also detect whether or not it was blocked, by simply consulting a real-time clock before and after every send. The elapsed time is not a foolproof indicator, but in many cases the resulting error rate over the covert channel will be quite low.

McCullough's example has several errors which are corrected in the following presentation. Two low-level processes A and B write to an output buffer BU visible to low-level users, and one high-level process, S, reads from one high-level input buffer BS. One of the three internal buffers, BC, is written by both A and B, and read by S. The other internal buffers, BA and BB, are written only by A or B, respectively, and are read by S. BA and BB have length one and block the sender when full. The other buffers may be of any length. The system is illustrated in figure 5.5.

Processes A and B each repeatedly execute four sends. A first sends two messages to BA, then sends a 0 to BU, and then sends to BC. B first sends two messages to BB, then sends a 1 to BU, and then sends to BC.

Process S repeatedly executes four receives. It first receives from BS, then receives twice from either BA or BB, depending on whether it received a 0 or a 1 from BS, respectively, and then receives from BC.

In operation, A and B fill up BA and BB and then block on their second sends. S reads a high-level bit from BS, unblocks one of A or B by reading from BA or BB, clears that buffer by reading again, and then blocks on BC, waiting for the unblocked low-level process to send to it. This gives the unblocked low-level process time to send its bit to BU, and it then allows S to continue by sending to BC.

The effect of this behaviour is that BU receives a bit-by-bit copy of the input to BS, clearly as a result of the blocking nature of the finite buffers in the system. Each individual process is secure from any standpoint; the low-level processes never *read*, and the high-level process never *writes*! Yet low-level users see a faithful reproduction of a high-level input sequence.

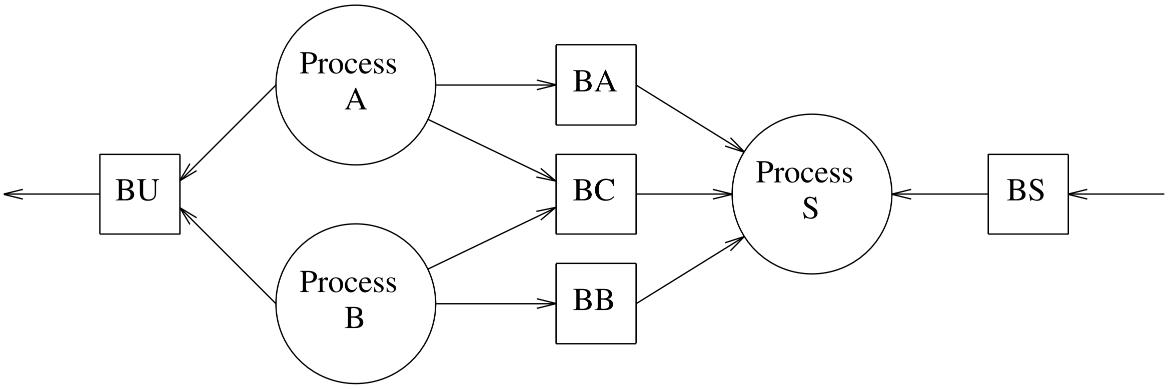


Figure 5.5: Covert Channel due to Buffering.

The remedy for this situation is to require buffers that are ‘‘effectively’’ infinite in capacity. If the probability of filling a buffer is very small, the amount of information leakable through this type of covert channel may be acceptable. This is treading on thin ice, however, for it is relying on details of the implementation to preserve security.

5.3.2. Generalized Noninterference

Goguen and Meseguer’s non-interference property is directly applicable only to deterministic systems. Since it has just been shown that this requirement is very restrictive (if not impossible to meet) in practice, McCullough presents a generalization of Goguen and Meseguer’s model. This is the definition of non-interference in terms of traces that was discussed in the earlier paper.

It was shown in that paper that this ‘‘generalized non-interference’’ is not a composable property. However, it is claimed that generalized non-interference is composable if there is no feedback between systems. Thus, systems with the generalized non-interference property can be connected as a directed, acyclic graph with infinite internal buffers to form a composite system with the generalized non-interference property. This result is clearly not as general as we would like.

5.3.3. Deducibility

Sutherland's deducibility property is applicable to non-deterministic systems in its original form, but it is even less composable than generalized non-interference, according to McCullough. However, if a deducibility-secure system makes no high-level output unsolicited by a high-level input, it is composable.

Unfortunately, the *unsolicited write-up* requirement is probably too restrictive in practice, since this forbids systems that simply upgrade the level of inputs. McCullough also claims, as in the earlier paper, that deducibility is not an adequate security property when applied to non-deterministic systems, but does so by way of an example that is nearly identical with that in the earlier paper. Again, this claim is well founded, but the inappropriateness of his examples reflects what appears to be an imperfect understanding of the deducibility notion.

5.3.4. Restrictiveness

Clearly, neither generalized non-interference nor deducibility strengthened with the unsolicited write-up requirement is an adequate composable security property in the presence of non-determinism.

To remedy this, McCullough had earlier introduced the ''hook-up security'' property as a requirement on the allowable traces of a machine. The closing paragraph of his 1987 paper states that a property on traces may be more suitable than one defined for state machines, as Goguen and Meseguer's work had been. It is interesting, then, to find his hook-up security property presented in 1988 as ''restrictiveness'', a property defined in terms of state machines.

That he would do so is, however, understandable, since applying the trace model becomes unwieldy for even the simplest real-world systems. This can easily be seen by applying the non-interference definition in terms of traces to any of the examples from the two papers.

A potential problem with defining security properties in terms of state machines, though, is that some generality may be lost. However, it is claimed that, for any set of *restrictive traces* (that is, a hook-up secure set of traces), there is some *restrictive state machine* that gives rise to only those traces. Conversely, all restrictive state machines produce only restrictive traces.

The transitions in McCullough's state machines are taken on input and output events. The set of possible traces for a machine is the set of event sequences produced by successive transitions starting from the initial state.

Two states are said to be *equivalent* if it is impossible for low-level users to distinguish between them. States which are equivalent can be grouped into *equivalence classes* ; all states in an equivalence class can be said to have the same *low-level state* .

For such a state machine to be restrictive, it must satisfy four conditions:

- The machine must be *input total* , which means that it must be permissible for any input to occur at any time. That is, there must be a transition from every state for all possible input events. In practice, this means that inputs must never be blocked; buffers will have to be large enough to cope with worst-case combinations of input and processing rates, or inputs to full buffers will have to be silently ignored.
- All states belonging to one equivalence class must go to states belonging to another equivalence class on the same low-level input. In other words, states with the same low-level portion must appear to respond identically to a low-level input.
- Transitions due to high-level inputs must go between equivalent states. This means that high-level inputs cannot directly affect what is visible to low-level users.
- All states belonging to one equivalence class must go to states belonging to another equivalence class on any high-level output sequence, possibly with an embedded low-level output. In other words, states with the same low-level portion must appear to behave the same regardless of the high-level outputs being produced. The nature of high-level outputs should not be evident from changes in the low-level state.

Hooking up two state machines A and B forms a composite machine whose states are pairs of the states of A and B. The state $\langle A_1, B_1 \rangle$ is equivalent to $\langle A_2, B_2 \rangle$ if and only if A_1 is equivalent to B_1 and A_2 is equivalent to B_2. Allowable transitions between composite states occur when either machine has a non-shared input or output event (in which case only one of the states in the composite pair changes), or when one machine's output is the other's input (in which case both states in the pair change).

It can be shown that this composite machine is restrictive (and satisfies the above four restrictiveness conditions) if the component machines are. The proof appears in ''The Theory of Security''.

5.4. SECURITY AND MACHINE COMPOSITION

We will now consider two contributions to the theory of security composition which were published in the Proceedings of the Security Foundations Workshop in 1988.

The first paper, *Security and the Composition of Machines* [Johnson88], by Dale M. Johnson and F. Javier Thayer, builds on the foundation provided by Darryl McCullough, whose work is discussed elsewhere in this report. Johnson and Thayer present a new security property that is similar to McCullough's, and has the same desirable quality of composability, but which is demonstrably weaker (i.e., less restrictive).

The second paper, *What Needs Securing?* [Guttman88], by Joshua Guttman and Mark E. Nadel, uses a different approach to the composability question, while continuing to characterize systems by their sets of possible execution traces. As may be expected from the title of this paper, it also considers the contentious issue of how confidentiality itself should be defined. Although this latter question is not directly related to composability, it is certainly of vital interest that the security condition met by a composed system be an effective and practical one, and so we shall consider this issue in some depth as well.

It is undoubtedly fair to say that McCullough's work was an effective catalyst in the creation of both these papers. McCullough, in turn, based his essential confidentiality condition on the non-deducibility definition of Sutherland. In recognition of their progenity, we will refer often to these works to show both their similarities to and diversity from the papers under examination.

5.4.1. Johnson and Thayer

No better introduction to this paper can be made than the following from its abstract:

> This paper examines definitions of security with respect to inputs in the context of abstract machines defined by means of concurrent event systems, and it identifies this area of security under the notion of *correctability* .

The first part of the paper introduces the *event system* formalism within which the balance of the paper is conducted. This is an abstract representation of a system employing a black box approach. Operation of the system is modelled as a sequence of *events* , which include input and output events, possibly in addition to other strictly internal events. Each possible, legal system execution sequence is called a *trace* , and the system is fully characterized by its set of possible events and its set of valid execution traces.

This approach is attractive from a theoretical viewpoint because it is capable of describing a system using only its externally visible behaviour. It is not necessary to make any assumptions about the internal operation of the system in order to model it. Contrast this with a modelling approach based on state machines, where the model is required to incorporate some approximation to the internal operation of the modelled system. This is less desirable because

- It represents unnecessary implementation-related detail. In the abstract model we are really interested in a system's behaviour, rather than in the details of its construction.

- The state machine formulation may implicitly constrain some of the behaviour of the model, and our subsequent analysis may depend on these constraints without our noticing it.

On the other hand, the trace characterisation has the major drawback that it is virtually impossible to work with in any concrete sense. An exhaustive list of all possible execution traces is not feasible for any but the most trivial systems, so some finite form of representation is necessary. This might take the form of either a decision procedure (predicate) that lets us decide which event sequences are valid traces, or a state machine whose handle can be turned to generate valid traces. Since actual implementations are usually based on state machines, concrete and application-specific models are quite appropriately constructed using them. However, at the most abstract modelling level, we prefer the event trace formalism.

It should be noted that, as in the case of most formal treatments of security, the scope of this paper is limited to relatively static versions of conventional (i.e. national security) confidentiality policies. It deals with systems each of whose inputs are marked with a *classification* , and whose outputs are similarly marked each with the *clearance* level of observers able to view that output. However, these inputs and outputs are not specific channels or physical ports, but are the (abstract) I/O events experienced by the system.

Each event is individually rated as to its classification or clearance, allowing any distinguishable pair of events to have different ratings. The internal structure of these abstract events is not addressed in the model, so events may be distinguished by any combination of characteristics including physical I/O port, the user on whose behalf the event occurs, or perhaps the particular data encapsulated in the event.

For this reason, it is not a strict requirement of this modelling scheme that the physical I/O channels of a system be single-level. It would be possible to apply the results obtained here, or in any similar model, to systems with multi-level interconnections. The danger in doing this is that the model would have to treat such a connection as a collection of isolated virtual single-level connections. The mechanisms that maintain separation of levels could not be represented in the model.

This is not a problem that is peculiar to this modelling scheme. It does, however, illustrate that an abstract security model cannot stand alone. It will always embody assumptions about detailed function that must be recognized and verified by more detailed analysis.

The authors next restrict the set of systems they will consider to those that satisfy two properties. They must have sets of traces which are *input-extensible* and *prefix-closed* . Input extensibility is, as the authors admit, a strong requirement. The system must be capable of accepting an unbounded string of inputs at the end of any given trace, before producing any output (or internal) events.

Prefix-closedness is a property that is consistent with our intuitive interpretation of an event sequence as a temporal ordering of events. If a system is able to execute a trace τ by a particular time, it makes sense that at some earlier time it would have only executed some prefix of τ. Despite this, it is not as trivial a restriction as it seems. If all initial segments of a trace are to be valid traces, then no trace may contain events that are inseparable.

The combination of input-extensibility and prefix-closedness leads us to a dilemma. On the one hand, no two consecutive events may be bound together so closely as to be inseparable. This might be interpreted as ruling out arbitrarily rapid input sequences, so that there must be some limit to the number of events that can occur in any finite span of time. On the other hand, the system must be capable of accepting a string of inputs of unbounded length before producing any outputs. It appears that the only way to satisfy both these properties at once is to require that the system be capable of delaying outputs an unbounded amount of time.

This is not a benign statement, as it might be if it were in an ordinary functional specification. A designer might be quite comfortable with a conventional specification that does not bound response time, because it would ordinarily be interpreted as a lack of a requirement, or as permission for the implementation to not meet any particular response time limit. In this case, however, it is a positive requirement, and any implementation that fails to occasionally delay outputs for very long periods could not be considered proven secure using this theory. McCullough's Hook-Up property has a similar difficulty.

A curious consequence of these requirements is that a simple length of wire, carrying high-level messages with a fixed propagation delay, cannot be a component of a composite system modelled using either McCullough's or Johnson and Thayer's formalism [1]. And, we cannot safely gloss over the problem, claiming some particular system is ''pretty close'' to conformance with this type of requirement, because the formal proof of security may hinge on one of these properties. We would prefer a formalism that managed to avoid these difficulties entirely.

This particular concern is related to the larger question of how much trust may be comfortably placed in formal security definitions. Too often, they seem to rely on theoretical approximations that ignore practical considerations such as the relative probabilities of different events. Proofs of security, on close examination, may hinge on the occurrence of circumstances that are possible but very unlikely. Development of the model in the TNA

[1] Components may contain wires — one presumes they interconnect somehow — but a wire may not by itself be a separate component.

report was pursued with a view to alleviating these concerns [2].

5.4.1.1. Formal Security Framework

More for the sake of definiteness than any other reason, the authors restrict their notion of confidentiality to one which prevents low-level users from deducing anything about high-level inputs. As they remark, the question of whether this *input security* is the ''right'' definition is not yet settled. This issue is addressed more thoroughly in the paper by Guttman and Nadel, and we will postpone further discussion until we deal with it.

The paper essentially uses the notation of McCullough, defining the *event trace system*

$$S = \langle E, I, O, T \rangle$$

where

E is the set of events

I, the input events, is a subset of E

O is a subset of E disjoint from I, and is the set of output events

$T \subseteq E^*$ is the set of traces

The set of events E is also divided into the disjoint subsets L and H, such that every event is in exactly one of L or H. These are, respectively, the sets of low- and high-level events. Assuming two comparable levels simplifies the discussion without any significant loss – the generalization to an arbitrary lattice of levels is straightforward but a little messy.

A more serious shortcoming is that the simple classification of events used here assumes that events are entirely input or output oriented. In practical systems this is rarely the case. Input and output operations usually include some reverse-flowing acknowledgment or handshaking information. The ability to handle composite input-output actions was present in the TNA model [3].

Throughout the rest of the paper these symbols are combined using standard set operations, yielding such combinations as the intersection $H \cap I$, the set of high-level input events, and $L \cup (H \setminus I)$, the set containing all low-level events plus all high-level non-input events. Also introduced here is the notation $\alpha | L$ which is the subsequence of sequence α containing only events in L.

---

[2] [Thomson88], p. 135.

[3] [Thomson88], pp. 152-153.

5.4.1.2. Correctability

The first important concept, that of *correctability*, is introduced on page 75. It is restated here in a slightly paraphrased form:

> An event system is *correctable* if and only if[4] for every trace $\alpha \in T$ and every sequence $\beta \in (L \cup (H \cap I))^*$ satisfying $\beta | L = \alpha | L$, there exists a trace β' such that $\beta' | (L \cup (H \cap I)) = \beta$.

In this definition, β is an arbitrary interleaving of the low-level behaviour of trace α together with any sequence of high-level inputs. If the system is correctable, there will exist a valid trace β' that contains the same interleaving. It will probably also contain high-level internal or output events, but these are not significant when confidentiality is defined as input security.

An alternative equivalent definition is provided in the paper. A *perturbation* of a trace is a sequence formed by inserting, modifying, or deleting high-level inputs in the trace. A *correction* of a sequence is a trace obtained by inserting, modifying, or deleting high-level non-inputs (i.e. outputs or internal events) in the sequence. A system is correctable if every perturbation has a correction.

5.4.1.3. Comparison with Deducibility Security

Correctability is similar to Sutherland's deducibility security, yet differs from it in important ways. To see this more clearly, we state a deducibility-security condition (for input security) using the same terminology:

> An event system is *deducibility-secure* iff for any two traces $\alpha, \gamma \in T$ there exists a trace $\lambda \in T$ such that $\lambda | L = \alpha | L$ and $\lambda | (H \cap I) = \gamma | (H \cap I)$.

The third trace λ exhibits the low-level behaviour of α together with the high-level-input behaviour of β.

The paper goes on to show that correctability is implied by McCullough's hook-up security property. We find it more instructive to compare correctability with non-deducibility. It is apparent that correctability is stronger than non-deducibility on two grounds:

1) Correctability provides that all sequences of high inputs are legal in combination with any observable low behaviour. Deducibility, on the other hand, only requires this if the high input sequence is already legal in some other trace. Note, however, that this difference disappears if we

[4] The original omitted the ''only if''. They could do this because they only attempted to prove that correctability was implied by other things, not that correctability implied anything else. In any case, it was probably a typo.

further specify input-extensibility and the input prefix property, because then we can always construct a legal empty trace then legally extend it with any desired sequence of high inputs.

2) Correctability requires that all *interleavings* of low activity and high inputs be possible. Deducibility security only promises that a trace exists with the desired subsequences, it does not guarantee anything about how they are interleaved.

The latter is the more significant difference. Whether or not it makes correctability better than non-deducibility as a definition of confidentiality is a difficult question. It might be considered better if we take the position that input security is not sufficient, and that the interleaving is itself sensitive information worthy of protection. This view is propounded by Guttman and Nadel, and we will consider it later in the discussion of their paper.

The other possibility is that, even without regard to the sensitivity of the interleaving, correctability may yield a more satisfying definition of security. Consider a system with the following property: Once it has received a single high-level input, it enters an operating mode where each low-level input evokes a low-level response containing a copy of the most recent high-level input. Queries occurring before the first high-level input are answered with randomly generated data.

This system is plainly capable of exhibiting any desired low-level behaviour when there are no high-level inputs. It follows that arbitrary low-level behaviour can be achieved in conjunction with any specified high-level input sequence, by simply requiring that all the high-level inputs occur after the last low-level input. We must conclude that this system is secure by the non-deducibility definition, since non-deducibility does permit us to pick such a ‘‘peculiar’’ interleaving in the composite trace.

Now, if we evaluate this system against the correctability criterion, it will be judged insecure. Correctability requires that the desired low-level behaviour be maintained across all interleavings of high-level inputs. In this instance, at least, correctability seems to agree more closely with our intuition than does deducibility.

Still, the subject is not necessarily closed with this observation. Deducibility is known to have problems dealing with non-deterministic systems, as discussed at some length in the TNA report [5]. Although correctability is sufficient to handle our current example, other non-deterministic scenarios exist in which correctability is no better than straight non-deducibility [6]. The solution recommended in the TNA work was elimination of the non-determinism in the system’s behaviour. If we assume this to be done in both

[5] [Thomson88], pp. 93-109.

[6] Eg. the example on p. 93 of [Thomson88].

cases, the question that remains is whether there is any significant difference between correctability and non-deducibility when restricted to deterministic systems.

We present here a non-rigorous [7] first cut at arguing this question. Begin by assuming the existence of a deterministic system S that is deducibility-secure but not correctable. S must have a trace α such that for some interleaving β of $\alpha|L$ with a high-level input sequence, there does not exist a trace β' satisfying $\beta'|(L \cup (H \cap I)) = \beta$.

Input-extensibility is incompatible with determinism [8] so we make the weaker assumption that for any input-only sequence there exists at least one trace containing those inputs. Let θ be such a trace with inputs $\theta|I = \beta|I$. Therefore, $\theta|(L \cap I) = \alpha|(L \cap I)$.

Now, if we apply the non-deducibility property (with γ equal to the empty trace), we find that there must exist high-input-free traces α' and θ' such that $\alpha'|L = \alpha|L$ and $\theta'|L = \theta|L$. But then, because of the way θ was constructed, θ' and α' must have the same low inputs. Since the system is strongly deterministic, they must then be the same trace $\alpha' = \theta'$. This, in turn, implies that $\alpha|L = \theta|L$, and it then follows that $\theta|L = \beta|L$.

Restating what we have so far:

$$\theta \in T$$

$$\theta|(L \cap I) \cup (L \cap O) = \beta|(L \cap I) \cup (L \cap O)$$

$$\theta|(L \cap I) \cup (H \cap I) = \beta|(L \cap I) \cup (H \cap I)$$

$$\theta|(L \cap I) \cup (L \cap O) \cup (H \cap I)) \neq \beta|(L \cap I) \cup (L \cap O) \cup (H \cap I) \quad .$$

The remaining difference must be in the relative ordering of the low outputs and high inputs. Further exploration of this difference is deferred.

5.4.1.4. Composition

Beginning on page 77, the authors introduce the notion of composition of event systems. Composition is formalized using the notation devised by McCullough. The definition is reproduced here:

---

[7] Here we use the strong definition of determinism that appeared in the TNA report. In fact, the definition is too strong, and a weaker one such as that used by McCullough in his *Theory of Security* should be more useful.

[8] Input extensibility implies that all sequences consisting only of input events are valid traces. Thus, for any trace α, $\beta = \alpha|I$ is also a trace. But α and β have the same inputs, so if the system is deterministic we must have $\alpha = \beta$, i.e. the system can never produce outputs.

An event system $S = \langle E, I, O, T \rangle$ is the *simple hook up* of component systems S_1 and S_2 if the following conditions hold:

$$\begin{aligned}
E &= E_1 \cup E_2 \\
I &= (I_1 \setminus O_2) \cup (I_2 \setminus O_1) \\
O &= (O_1 \setminus I_2) \cup (O_2 \setminus I_1) \\
I_1 \cap I_2 &= \varnothing \\
O_1 \cap O_2 &= \varnothing \\
(E_1 \setminus (I_1 \cup O_1)) \cap E_2 &= \varnothing \\
(E_2 \setminus (I_2 \cup O_2)) \cap E_1 &= \varnothing
\end{aligned}$$

and $T = \{\alpha \in E^* \text{ such that } \alpha | E_1 \in T_1 \wedge \alpha | E_2 \in T_2\}$

What this means is that the events of the two component subsystems are defined in an overlapping fashion, such that some events are simultaneously outputs of one system and inputs of the other. The I and O sets of the composite are the external inputs and outputs of the connected system.

We may point out here that there is a slight notational obstacle in using this hook-up definition. When we prove things about hooked-up components, we would like to be able to make statements of the form ‘‘For all legal hook-ups of S_1 and S_2, property X holds’’. However, in the definition above, the hook-up is implied by the definition of the subsystem event sets. Given any particular S_1 and S_2, there is only one way they can possibly be hooked up.

For the purists, the statements we make have to be modified replacing ‘‘instances of S_i’’ by ‘‘subsystem isomorphic to S_i’’. For the rest of us, the notational simplicity provided by this definition greatly exceeds the significance of any problem it introduces, so we are content to use it without modification.

5.4.1.5. Forward Correctability

Having settled on correctability as their definition of confidentiality, the authors then remark that it is not preserved by composition. They propose a restricted form, called *forward correctability*, which is a composable property. Following is a concise definition of their most generalized form of this:

> Let $\alpha = \beta \iota [i]^n \gamma$ be a trace of S, where $\gamma \in (E \setminus H \cap I)^*$ contains no high-level inputs, ι is a sequence of high-level inputs, and where $[i]^n$ denotes a (possibly empty) sequence of low-level inputs of length no greater than n.
>
> Iff, for any such α, and for any alternate sequence ι' of high-level inputs, there exists a $\gamma' \in (E \setminus H \cap I)^*$ satisfying $\beta \iota' [i]^n \gamma' \in T$, then S is *n-forward correctable*.

If S is n-forward correctable for all $n \geq 1$, it is said to be ω-*forward correctable* [9].

This rather involved definition turns out to be very similar to simple correctability, with these two modifications:

- it must be possible to correct every perturbation by modifying the sequence only after the perturbed part (if at all)
- moreover, if the perturbation happens to be immediately followed by n or fewer low-level inputs, it must be possible to delay any correction until after those inputs

There is a subtle difference between these two qualities. The first can be viewed as a fix for a basic weakness in the correctability definition of confidentiality. It yields something we might call *0-forward correctability* , or perhaps *causal correctability* , that is the first step toward providing a useful definition of composable security.

To understand this, we must first recognize the debt that correctability owes to the non-deducibility definition of security: the notion that a low user should not be able to deduce, through observations of low-level activity, anything about high-level inputs. Correctability, as we have seen, extends this protection even to the interleaving of high inputs with low-level events.

From this realization, it is only a small step to the perturbation-correction formulation of correctability: if the sequence of high inputs is altered (a perturbation), there must exist an execution (i.e., a correction) combining the old low-level behaviour with the new high inputs. McCullough recognized this and called it ''fixing up'' the modified trace[10] , though he did not christen the property of ''fix-up-ability'' with a name. He also saw a flaw: that some systems may be defined such that a correction sometimes has to occur before the perturbation.

McCullough calls this characteristic a violation of causality, which is somewhat misleading because it implies that such a system could not be built. The actual consequence, however, is more disturbing than this: the systems can be built but will not be secure.

Consider the following example: a system has one each of high-level input and output and low-level input and output. Every low-level input prompts the emission of a low-level output some time later. This output will have the value ''1''iff an equal number of high inputs and outputs occur between the low input and output and the last high event (if any) in that interval is an input, otherwise it takes the value ''0''. Some legal traces are shown in figure 5.6.

---

[9] ω is conventionally used in number theory to represent simple (i.e. countable) infinity.

[10] [McCullough88a], p. 65.

Trace A produces a ‘‘1’’output because there are no high-level events between the low-level input and output. A′ is a perturbation of A made by interleaving a high-level input between the low-level events. This change would normally force the low output to be a ‘‘0’’,as in trace B, but if the system is correctable there should exist a correction that will still produce a ‘‘1’’ output. Trace C is the desired correction.

Although this system is correctable, and satisfies the prohibition on low-level observations allowing deduction of high-level information, it can still be used to transmit sensitive information to a low-level observer. Its fault is that a malicious high-level user (eg. a Trojan horse) can wait until the system receives a low input, then can transmit one bit of information to a low-level confederate. To transmit a ‘‘0’’,send a high-level input, unless a high-level output has already occurred, in which case do nothing. To transmit a ‘‘1’’,do nothing, unless a high-level output has already occurred, in which case send a high-level input[11] .

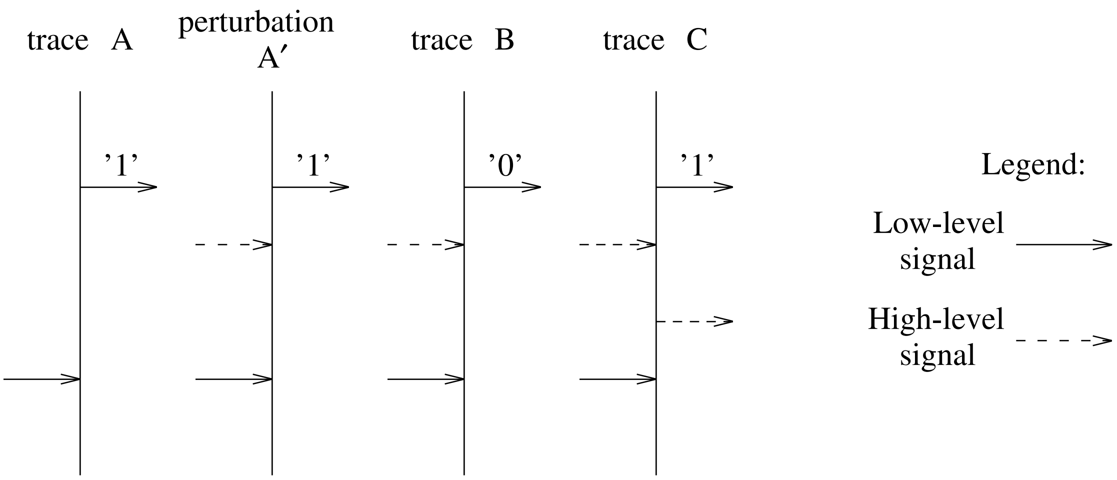


Figure 5.6: Possible Traces in Example Correctable System.

[11] The system may now produce another high-level output, requiring the spy to counter with another high-level input. It might be argued that the possibility of a running battle of high-level messages prevents the spy from reliably transmitting his bit. It is counter-argued that the spy may already reliably send a 0 bit, and that a slightly more complex system (in which high outputs following the high input are ignored) allows for reliable 1 transmission too.

The astute reader will recognize that there is a hook-up quality to this example, in that the spy is using high-level outputs to determine which high-level inputs to supply. High outputs, which should be ‘‘fixing up’’ a trace, can be defeated by the spy by manipulation of subsequent high inputs. The weakness is that the trace cannot be ‘‘fixed up’’ after the compromising input.

Feedback of this sort is a common feature of hookup-insecure systems. In the TNA report [12], we presented an example whose messages were sequences of levels rather than events. McCullough has an example [13] where the feedback loop contains two nontrivial subsystems. In each case, the system’s fix-up attempt that might spoil an opportunity to compromise information is thwarted by subsequent high inputs.

So, it is apparent that a system may be subject to this kind of abuse if it contains even a single trace for which a perturbation must be fixed-up in advance of the disturbance. If we require that all perturbations have at least one causal fix-up, it would appear that this vicious cycle could be broken, and that we might have a composable definition of security.

However, it turns out that causal correctability is not, in itself, sufficient to guarantee composability. The explanation why this is so will be postponed until we consider the proof of composability that appears later in the paper, because the problem that may arise is quite visible there.

It is worth emphasising that none of the correctability definitions (nor, for that matter, the non-deducibility ones) *require* that a perturbed system respond with a correction, only that it be *possible* for it to do so. It is therefore not guaranteed that a perturbation will have no effect on observable low activity, only that it will have no reliable effect. This distinction is part and parcel of the fact that the systems considered are not required to be deterministic. Applicability to non-deterministic systems is a vital difference between non-deducibility and Goguen and Meseguer’s non-interference, though not the only difference [14].

5.4.1.6. Comparison with Restrictiveness

With all of the parallels between forward correctability and McCullough’s restrictiveness that have been apparent so far in this report, a question that suggests itself rather strongly is whether there is any real difference between them, or whether they are the same notions expressed using different notation.

---

[12] [Thomson88], p. 107.

[13] [McCullough87] p. 165.

[14] [Thomson88], p. 98.

Such a comparison is complicated by the fact that McCullough defines his property twice, once relying on English [15] and again using a more precise mathematical expression [16], and the two definitions do not agree. However, further exposition in the first article cited [17] indicates that the textual description mis-states the author's intention, and that the more formal definition is the intended one.

Johnson and Thayer perform the comparison that interests us on page 80 of their paper, using the correct restrictiveness definition. They show that hook-up security is equivalent to ω-forward correctability (i.e. unbounded forward correctability), in the sense that a system that has one property also has the other.

Closer examination of the definitions, however, reveals that the two properties are equivalent not only in effect but also in formulation. Both definitions state that a perturbation occuring before an arbitrarily long sequence of inputs and a high-input-free final segment must have a correction delayed past the inputs. Restrictiveness and ω-forward correctability are not simply equivalent properties, they are actually the same property couched in different symbols.

5.4.1.7. Limited Forward Correctability

Johnson and Thayer's development of ω-forward correctability, although it does eventually result in the rediscovery of restrictiveness, makes the significant contribution of identifying the classes of n-forward correctable systems for finite n. These are of interest because, as demonstrated in the paper, they are strictly weaker properties than ω-forward correctability [18], and yet they are also composable properties. It is of interest to us to identify the weakest possible condition that guarantees composability.

Like ω-forward correctability (or restrictiveness), 1-forward correctability is composable, implies simple correctability (and, hence, deducibility-

---

[15] [McCullough87], p. 165.

[16] [McCullough88a], p. 85.

[17] The text in question is

> ... such that the first changed output of τ_3 occurs no sooner than the first output in the changed portion of τ_2.

which requires that the correction be delayable not only past the full sequence of inputs, but also past the first output following them. The restatement in the following paragraph makes it clear that the author did not really have such a delay in mind. This should be taken as an example of the dangers of using English as a specification language.

[18] I.e. the class of 1-forward correctable systems is a proper superset of the ω-forward correctable systems.

security), and is causal, but enjoys the added advantage of being weaker than n-forward correctability for any $n \in \{2, 3, \ldots, \omega\}$. 0-forward correctability is an even weaker requirement and has most of these properties but, as we indicated earlier, it is not composable. 1-forward correctability is therefore the weakest composable condition described so far in this paper.

It appears that the critical requirement for composability, at least among forward correctable systems, is that if the perturbation is immediately followed by a low input it must be possible to delay the correction until after that input. We can learn why this delay is necessary by studying the composability proof presented in the paper.

5.4.1.8. Proof of Composability

The proof presented on pages 84-87 of the paper is based on McCullough's composability proof for the restrictiveness property [19] but is organized differently.

We will reuse the *simple hook up* notation introduced earlier, as do the authors, and add for convenience

$$C = (I_1 \cap O_2) \cup (O_2 \cap I_1) \ .$$

This symbol represents the *system communication events* of the newly-composed system. That is, they are the output events of one component that are fed to the inputs of the other. They are, therefore, internal events of the composite system but inputs and outputs, respectively, of the components.

5.4.1.8.1. Lemma 1

Lemma 1 in the paper establishes that, under some constraints, traces of the component systems may be knitted together to form a valid trace of the composite system. We will dissect, not Lemma 1, but Lemma 1′ which follows, which has been slightly rephrased in the interest of even further simplification and clarity.

Lemma 1′

Let $\alpha \in T$ and $\tau_i \in E_i^*$ for $i = 1, 2$ be such that $(\alpha | E_i)\tau_i \in T_i$. Suppose in addition that $\lambda \in L^*$ is such that

$$\lambda | E_i = \tau_i | L \qquad \text{for } i = 1, 2 \tag{1}$$

Then there is an event sequence τ such that $\alpha\tau \in T$,

---

[19] [McCullough88a], section 9.6.

$\tau | L = \lambda$ **(2)**

and

$$(\tau | E_i) | (L \cup (H \cap I)) = \tau | (L \cup (H \cap I)) \qquad \text{for } i = 1, 2 \tag{3}$$

(Note that the I in the last equation represents *system* inputs, not component inputs, so excludes communication signals.)

In this lemma, we have:

α a trace of the composite system that is going to be extended. The lemma actually talks about knitting together extensions of traces, rather than whole traces, but the distinction is not really important.

τ_i are possible extensions to the execution traces of the two component subsystems. They are not quite independent because of the existence of ...

λ This sequence represents the target low-level behaviour of the composite system. It therefore contains all low-level events that the subsystems will experience, plus it specifies how they are to be interleaved in the composite system execution. Since it contains all low-level events, including low-level communication events, it restricts the τ_i extensions to ones that have mutually compatible low-level communication events.

One way to approach these is by imagining that the composite system temporarily has its high-level interconnections severed and reconnected to external sources and sinks. This *partly-hooked-up* system has all its low-level interconnections in place, but none of the high-level ones. The τ_i represent the achievable executions of the two component subsystems when they are partly hooked up in this fashion. λ then represents the interleaved low-level behaviour of the partly-hooked-up composite system. It contains the low-level events of the two component executions interleaved arbitrarily, except for the requirement that the two executions synchronise every time they execute a low-level communications event.

In the lemma statement, α represents the first part of fully hooked-up system execution. The τ_i represent partly-hooked-up extensions to this execution. It is as though the system executed the trace α while fully hooked up, and then the high-level interconnections were severed.

The lemma's assertion is that, if we do this and so determine a λ, that we can subsequently reconnect the high-level interconnections and achieve exactly that λ low-level behaviour of the composite system, by supplying the same system inputs (low and high) as before. This is true despite the fact that the high-level communication inputs, which previously were externally specified, are now supplied to each component system by communication outputs of the other component.

5.4.1.8.2. Lemma 1 Proof: Preliminary Step

The first assertion in this proof is that it is sufficient to consider only those τ_i that do not contain any high-level communication inputs. In terms of our visualisation of a partly-hooked-up composite, it means that for any τ_1, τ_2, and corresponding λ produced by the partly-hooked-up system, it can also produce τ'_1, τ'_2, and the identical λ by supplying only the system inputs (high and low) without any high-level communication inputs. When the system is then fully hooked-up, any composite execution τ that satisfies the stated conditions for the τ'_i must also satisfy them for the original τ_i, since they contain the same low-level events and high-level system input events.

In formal terms, given τ_i and λ satisfying equation (1), let τ'_i be the corrections of the perturbations formed by deleting all $H \cap C \cap I_i$ (high-level communication input) events. These corrections must exist since the component systems are correctable; the corrections are limited to the trace extensions (i.e. do not reach back into the basal trace α) because they are causal correctable.

Then, we must have

$$\tau'_i \mid (L \cup (H \cap I)) = \tau_i \mid (L \cup (H \cap I))$$

because none of those events were affected when forming the perturbation or when making the correction. It now follows easily that a τ satisfying equations (2) and (3) with respect to the τ'_i will also satisfy those equations with respect to the original τ_i.

5.4.1.8.3. Lemma 1 Proof: Base Step

Assume now that τ_i refer to sequences that do not contain high-level communications inputs. The lemma is first asserted for the case where there are no low-level events in the component traces, so the length of λ is zero. A composite trace extension τ is formed by concatenating the high-level system input subsequences of the τ_i,

$$\tau = (\tau_1 \mid (H \cap I))\,(\tau_2 \mid (H \cap I))$$

and relying on input extensibility to guarantee that $\alpha\tau$ is a trace.

It is unnerving to have a proof of security depend on a property that is as contentious as input extensibility. It should be a goal of our future research to remove this assumption as much as possible, or at least replace it with a more palatable one.

5.4.1.8.4. Lemma 1 Proof: Induction Step

Induction in this proof operates by showing that the desired properties are maintained for the first event e in the sequence $\lambda = e\,\lambda'$, then using the inductive assumption to take care of the tail λ'. In effect, events are removed from the beginning of one or both τ_i traces and knitted together extending the basal trace α until a low-level event is moved.

Four subcases are identified depending on the nature of event e. The one presented is the most complex, where e is a low-level communications event.

Without loss of generality, it is assumed that e is an output event of subsystem 2 and an input event of subsystem 1. Since e must be the first low-level event in both τ_i, we may write

$$\tau_i = \beta_i \, e \, \rho_i$$

where β_i contains no low-level events and, by previous assumption, no high-level communications input events.

It may, however, contain high-level communications output events. If so, when we knit the two β_i together to form a composite trace, those outputs will become high-level communications inputs to the other subsystem. Those new inputs are a perturbation and, by causal correctability, must have a correction that does not involve undoing any previous knitting.

The essence of composability, and of 1-forward correctability, is to ensure that this knitting process can be consummated. It is not difficult to imagine a process in which, as each output is fed from one subsystem to the other, the resulting correction requires that a barrage of new high-level communications signals be immediately returned to the first subsystem. In such a situation we might be indefinitely postponed from ever reaching (and consuming) the low-level event e, and so effectively prevented from successfully knitting together the subsystem traces. As we will see, 1-forward correctability is the key to preventing this situation.

We perturb $(\alpha|E_2)\tau_2$ by inserting the high-level communication events $\beta_1|E_2$, as follows:

$$\alpha|E_2 \, \beta_1|E_2 \, \beta_2 \, e \, \rho_2$$

This perturbation must have a correction

$$(\alpha\beta_1)|E_2 \, \beta'_2 \, e \, \rho'_2$$

the existence of which depends only on the causal correctability property. This accounts for all the communication outputs of β_1, but has not taken care of communication outputs in β'_2, and indeed may have created new ones. We now feed those to subsystem 1, perturbing $(\alpha|E_1)\tau_1$ by inserting the high-level communication events $\beta'_2|E_1$ to form

$$\alpha|E_1 \, \beta_1 \, \beta'_2|E_1 \, e \, \rho_1$$

This is also a perturbation, and must have a correction

$$(\alpha\beta_1\beta'_2)|E_1 \, e \, \rho'_1$$

Since e is a low-level input to subsystem 1, we are guaranteed by the 1-forward correctability property that any corrections necessitated by the insertion of $\beta'_2|E_1$ may be delayed past e. We do not have to go back and feed new outputs to subsystem 2, and so have successfully knitted a new basal trace

$$\alpha' = \alpha\beta_1\beta'_2 e$$

It should be noted that the other 3 subcases, where e is a system input or output event or an internal event of one subsystem, are not interesting because there need be no interaction between the subsystems. This is because event e appears only in one of the two extensions τ_i, and so we can move all of $\beta_i e$ into α immediately without trouble as long as the other subsystem is at least causally correctable.

5.4.1.9. Proof of Hook-Up Theorem

With the lemma taken care of, the actual hook-up proof is quite simple. We must show that the composed system is 1-forward correctable. Let $\theta\iota[i]\gamma$ be a trace of S, where $\gamma \in (E \setminus H \cap I)^*$ contains no high-level system inputs, ι is a sequence of high-level system inputs, and where $[i]$ is an optional low-level input. Also let ι' be an alternate sequence of high-level system inputs.

Consider the case where ι and ι' are at most one event long. Let e denote the event that is different, inserted, or deleted between ι and ι'. Without loss of generality, assume that e is an input to subsystem 1. Then, sequence $(\theta\iota'[i]\gamma)|E_1$ is a perturbation of $(\theta\iota[i]\gamma)|E_1$ which is known to be a trace of S_1.

It follows that there exists a correction $(\theta\iota'[i])|E_1\gamma'_1$ which is also a trace of S_1. Note that if i is an event of S_1 we must invoke 1-forward correctability of S_1, otherwise causal correctability is sufficient. This is required not because we are trying to produce a secure composed system, but because we are specifically trying to produce a 1-forward correctable composed system. In other words, if causal correctability had been sufficient to prove the lemma we would be able to get by with it here, too.

Meanwhile, the other subsystem S_2 is not affected by this changed input, and $(\theta\iota'[i]\gamma)|E_2$ is identical to $(\theta\iota[i]\gamma)|E_2$, so is a trace of S_2.

Now we invoke the lemma with instantiations

$$\alpha = \theta\iota'[i]$$

$$\tau_1 = \gamma'_1$$

$$\tau_2 = \gamma|E_2$$

and discover that there exists a system trace $\theta\iota'[i]\gamma'$ in which γ' contains no high-level system inputs. This is known because γ did not contain any of them, and because neither the formation of γ'_1 nor the use of the lemma introduced new ones.

5.4.1.10. Security Condition A

It is instructive to return to page 81 of the paper where the authors perform a slight detour, introducing a ‘‘straw man’’ alternative security definition which they claim is composable. This definition, called *security condition* **A**, is a form of forward correctability in which the correction must be delayable past the first low-level event after the perturbation.

It is not clear from this English description what should happen if there is no low-level event after the perturbation, but a subsequent example shows the authors’ intention to be that no correction should be necessary in that instance. This is consistent with the interpretation that the first low-level event may happen some time in the future, so the system must be willing to delay the correction past the current end of the trace.

This definition fits our requirements for a ‘‘good’’ composable non-disclosure property, since it is correctable, composable, and causal. It is not immediately clear, though, how it compares with the hierarchy of n-forward correctable systems. According to security condition **A**, a correction must be delayable past a single low-level event, which need not be an input, as well as past any high-level events that intervene between the perturbation and the low-level event. In contrast, the 1-forward correctable systems delay past, at most, a single low-level *input* immediately following the perturbation. From this we can conclude that condition **A** is stronger than 1-forward correctability, since **A** implies 1-forward correctable but the converse is not true.

This means we need not consider condition **A** any further. As a restriction of 1-forward correctability, it does not introduce any new, useful concepts, and since it is stronger it is not competitive in the race for weakest-composable-security-condition honours. All this is fortunate since, as the paper points out, condition **A** requires behaviour that is not compatible with the provision of useful and productive data processing services.

5.4.1.11. Observations

From our study of this paper, and our in-depth analysis of the composability proof that it contains, we see that composability of confidentiality can be approached intuitively in the following manner:

1) Begin with some adequate trace-based definition of confidentiality. In this paper, the choice made was 0-forward correctability, a causal form of non-deducibility in which even the interleaving of high and low inputs is protected. This definition should have the property that, for each of the component machines, no low-level input sequences can cause compromise of high-level information.

2) Consider the partly-hooked-up system, in which low-level interconnections are made but high-level ones are left open. This configuration should still be secure, because sensitive information is still confined to

individual component systems and we have assumed that no manipulation of low-level inputs can cause compromise. Feed the desired low and high level system inputs into the system and observe the low-level behaviour.

3) Now make the high-level interconnections, too, and then show that the low-level behaviour that was previously observed can still be achieved. This may involve adding new constraints to the component systems. In this paper, 1-forward correctability served this purpose. Since low-level observers see exactly what they saw before, and those observations were known to not constitute a compromise, the fully hooked-up system must also be secure.

We anticipate that a similar approach may prove fruitful in extending composability to other aspects of security. In any case, this analysis has yielded valuable insights into the functioning of the McCullough/Johnson&Thayer composability theories.

5.4.1.11.1. Proofs of Other n-forward Composability

If we take the 1-forward correctability hook-up proof and replace $[i]$ with $[i]^n$, and assume n-forward correctability in the component systems, we will have a proof of composability of n-forward correctability. Note that the resulting proof uses Lemma 1 as is, not some strengthened version of it, because the $[i]^n$ inputs are shifted into the basal trace before the lemma is invoked. Thus, 1-forward correctability is still sufficient to perform the knitting operation.

This observation reinforces our idea that the 1-forward correctability is the essential ingredient that allows the creation of a mutually compatible fully-hooked-up trace from two partly-hooked-up ones. It seems clear that any related property is composable, as long as that property implies 1-forward correctability.

5.4.1.11.2. Alternative Lemmas

The Lemma's knitting-together process, which is fundamental to the composition proof, depends on the fact that the corrections resulting from feeding a high-level communications output from one subsystem into the other are not always required to occur immediately, so that progress of the knitting is not forced to be indefinitely postponed. It still may be postponed, and perhaps for a very long time, because the subsystems are not *required* to delay their corrections. However, this is not a security problem. Even when uncoupled, subsystems are not even required to always perform corrections. Their security rests on the uncertainty of whether or not a correction occurred, which prevents the unauthorized deduction of sensitive information.

In the Lemma, knitting-together progress is made possible by requiring that corrections be delayable until after a low-level input. This clearly works, and makes the proof simple, but it is also clear that there exist alternate rules that would accomplish the same end.

- Correction must be delayable until after a single low-level *communications* input.

If we know that some inputs of a subsystem are never going to be hooked up to another multilevel system, there is no need to constrain them in the same way.

- Events in the correction that are *new communications outputs* must be delayable until after a single low-level input.

Similar reasoning to the previous case.

- Correction must be delayable until after a single low-level *output* .

It is a simple exercise to modify the lemma proof to work with this constraint (just interleave the communications events as $\alpha \beta_2 \beta'_1$).

Further consideration of these and other possible rules is left for future work.

5.4.2. Guttman and Nadel: What Needs Securing?

This paper takes quite a different approach to the composition of security than was visible in the work of McCullough and of Johnson and Thayer. However, the composition question is not the only question the authors choose to deal with. They take the quite reasonable position that one must first decide which formal definition of security (non-disclosure) is appropriate before tackling the question of how to assure its maintenance under composition. Since we agree that these are the correct priorities, we will begin by responding to their concerns about the nature of non-disclosure.

The authors make the point that the examples they construct are didactic rather than pragmatic, and should be considered on the basis of the points they raise about formal security modelling. We will respect this caveat, and try to limit our comments to the illustrated policy issues rather than the possibly off-beat nature of the example.

5.4.2.1. Sensitivity of Output

The introductory discussion on page 36 of the paper establishes that the authors accept as a starting point for their work a non-deducibility definition of non-disclosure. They agree that the objective is to prevent low-level observers from inferring previously unknown and proscribed information about high-level activities. They then diverge from previous practice, asserting that it is not necessarily sufficient to protect only high-level inputs from discovery. In particular, the opinion is offered that high-level outputs may also be innately sensitive.

5.4.2.1.1. Divergence from Previous Work

Researchers attempting to create a formal definition for confidentiality have traditionally concentrated on hiding high-level inputs. Our understanding of this practice is that it is based on a deterministic model of data processing. A system receives both high and low-level inputs and produces outputs that are made available to high and low-level users. However, the so-called high-level outputs may validly depend on non-sensitive information. Since low-level users may know this information, any definition of security that required them to have no knowledge of high-level outputs would be unworkable[Sutherland86]. Instead, it is postulated that the sensitivity of the high-level outputs derives from their dependence on previous high-level system inputs. That is, the inputs are sensitive by definition, but the outputs are only so if their contents furnish some clue to the inputs. We may say that those inputs are *innately* sensitive, and that the dependent outputs are *consequentially* sensitive. It is precisely the function of a good security model to analyze system operation and determine which outputs may contain sensitive information. The modeller may then determine whether the actual sensitivity of each output is well matched to its audience.

Guttman and Nadel appear to disagree with this proposition at a fundamental level. They appear to believe that the high-level system *outputs* are the signals that are innately sensitive. They give their own putative explanation for the prevalent custom of hiding inputs: that by doing so, researchers believe that they will succeed in adequately protecting outputs, which was their real intention all along.

5.4.2.1.2. Examples of Output Sensitivity

Regardless of whether sensitivity is inherited by outputs from inputs or the other way around, the authors do raise an important issue when they say

> the notion that constraints propagate backwards evaporates in the presence of nondeterminacy.

They present in their Example 1 a system which autonomously generates high-level outputs. Clearly, any sensitivity of these outputs need not derive from any high-level inputs. Indeed, the system need not accept any inputs at all (eg. a key generator unit) and its outputs may be sensitive not because of their information content but because of the purpose for which they will be used.

The authors also present other examples which, curiously, seem less convincing than the first. Example 2 is a system that accepts low-level votes as inputs and, when a fixed number have been received, outputs a high-level decision. It is argued that even if the voters reveal their votes to one another, together with the time each was submitted, uncertainty in the time that they arrived at the processor would make it impossible for them to deduce the decision. The weakness we find here is that, since the inputs are low, there appears to be no policy barring the users from knowing when they did arrive. The system might quite legally acknowledge each vote with a message indicating the exact time the vote was accepted. We would argue that here the difficulty is not one of nondeterminacy, but one of upgrading, which requires and deserves separate consideration.

Example 3 contains a decryptor, and it is argued that the output is sensitive although the inputs are not. This is clearly not the case, since one very important input is the decryption key, which is very sensitive indeed. In this case, at least, the conventional approach will suffice. It does not deal as well with the inverse process of encryption [20], but innate output sensitivity will not help there.

The fourth example proposes a sensitive database query system. It is suggested that it may be permissible for low-level users to know what queries are issued by the high-level users, but not to know what outputs are produced in response. It is clear that a Trojan horse would flourish in a pasture such as this. We would not be comfortable with a modelling scheme that allowed this system to be branded secure without a fight. Something else should be present in the model — either the high-level users are specially vetted **trusted subjects** in the Bell and LaPadula sense [21], or there is a sanitizing filter at work between their inputs and the outputs to the low users. Meanwhile, the contents of the high-level outputs should be adequately protected by conventional models since those outputs depend on high-level inputs supplied when the database was being loaded.

5.4.2.1.3. Should Outputs be Sensitive?

Guttman and Nadel's use of output sensitivity affects two situations: *upgrading* and *non-determinacy*. In both cases, a system may appear to invent outputs that are more sensitive than the inputs it consumes.

---

[20] Particularly public-key encryption, where both the output and the key are non-sensitive. The sensitive data input is deducible, but this deduction is so difficult that it is effectively impossible. Cryptologists have successfully formalized notions of *difficult* and *effectively impossible*, but systems security researchers have not yet integrated these ideas into their work.

[21] In which case they are actually multilevel subjects, receiving high signals and producing low ones. Note that a composable definition of security might be useful when defining how they should behave.

Upgrading, or writing up, is not a problem for a conventional security definition, but it becomes a problem for the authors because of their requirement that outputs marked *high* be kept secret. This is shown in their example 8 on page 41, where a composite system produces the same signals on its low-level and high-level outputs. We argue that the problem here is not with the example, but with the requirement, since it is clear from the construction of the system that the information is not sensitive. It simply must be acknowledged that the label on an output is only an upper bound for the sensitivity of the information that may appear on that port. In the example, low-level data appears on an output marked ''high'', and a security model should not take exception to it.

Non-determinacy is a more serious problem. A non-deterministic system adds information to its outputs in addition to that (if any) inherited from inputs. It is true that, depending on how it is used, this added information may need to be protected. However, Guttman and Nadel's resolution of this question is to assume that any non-deterministic effects visible at a high-level output must be sensitive.

Our view is that such a policy is unnecessarily harsh. If high-level outputs are permitted to depend on low-level inputs, we believe that those outputs should also be able to depend on non-determinacy that is not sensitive. A suitable solution would be one in which the system definition includes an explicit classification of the non-deterministic effects.

Just such an approach was suggested in the TNA report. There, non-deterministic systems were modelled by representing them as a deterministic system with an extra random input that explicitly accounted for non-determinacy. This input was required to be assigned a sensitivity label, just as like all other inputs. It is a conscious decision of the modeller what level to use. Explicit representation of this effect also helps resolve other modelling problems, as related in the report.

One last practical consideration is how to determine the correct label for the newly introduced input. It is true in some of these cases that the sensitivity propagates backward from the output to the source, since it depends on the use to which the output is put. For these rare cases, it would be useful if the modelling scheme permitted an *a priori* sensitivity to be attached to selected aspects of the output. The model will then verify that the random input's label was set correctly. The TNA model had this capability, because it could specify which parts of each high-level event were to be considered innately sensitive.

5.4.3. Sensitivity of Interleaving

On page 39 of their paper the authors raise a second objection to previous formal security definitions. Previous work [22] has largely been concerned with ensuring that the sequence of high-level inputs is not deducible by low-level observers. Similar efforts have not been made to protect the interleaving of high inputs with low activity.

Such definitions allow systems to be branded secure when they permit deductions about interleavings as long as the actual high-level input sequences are not knowable. The authors give two examples. In the first, a low-level user wants to know when a high-level supervisor is monitoring his activity. However, it is not obvious how the system could usefully provide that information without also indicating whether or not the high-level input had occurred, or how many times it had occurred.

The second example is a little more thoroughly described, and we are able to comment more meaningfully on it. There is a high-level input consisting of a sequence of digits, and also a low-level input. If the low input occurs after some high inputs, then all succeeding high inputs are echoed to the low user. If the low input occurs before any high inputs, a random sequence of digits is echoed and succeeding high inputs are not echoed.

A deducibility approach will call this system secure if, for any combination of high inputs and low activity, there exists at least one innocuous interleaving of the two. In this example, it may always be assumed that the high inputs (if any) occur after the low input, and in that way any pair of sequences may coexist. The low observer cannot deduce that any particular high input sequence is impossible, but he can deduce that certain interleavings of high inputs with his low input could not have happened.

There are two questions that need answering here. One asks whether this is really a new problem, or simply a new manifestation of an old one. It is easy to invent scenarios like these, for which deducibility appears to give wrong answers, but the examples always have trouble dealing with the boundary case where there are no high inputs. Satisfying the deducibility criterion for that situation seems to require resorting to non-determinacy, as in the example above. We are already aware that non-determinacy can lead to bad results, and if they are an essential part of this bad result then we would be better off tackling the root of the problem rather than one of its branches.

If we are unwilling or unable to make that determination, the second question to consider is whether what is leaked is actually sensitive. It appears from these examples, and from others like them, that although no deductions can be made about the entire high-level input sequence, some information is available about parts of it. Instead of learning about all high inputs, the observer may learn about high inputs that occurred more than an hour ago, or only within the last six hours, or since the last low input. Since we can make no assumptions about how sensitivity varies with time, we must

---

[22] Specifically, the non-interference model of Goguen and Meseguer and the non-deducibility model of Sutherland. McCullough's and Johnson and Thayer's security definitions do protect interleaving.

conclude that this information is indeed sensitive.

Whether specific efforts are needed to protect interleaving is not a question that we can answer at this early stage. It may be that meeting the already recognized need to handle non-determinacy will also eliminate this problem. Further study is indicated here.

5.4.4. Hook-Up

Having decided that security should include protection of high outputs and of interleaving, the authors next begin to develop a composable security definition that has these properties. They make the twin points that high signals should be protected as an aggregate, and that low observers should be assumed to have access to all low-level information. These are reasonable and accepted principles.

The formal framework used is similar to that of the previous paper. Event traces characterise the operation of the system, and the prefix-closedness and input-extensibility properties are assumed. The notation used to describe the interconnection involves a hook-up function $h(c)$. Explicitly representing the hook-up strikes us as a positive step, since it at least makes it possible to state assertions quantified over hookups.

Another difference between this paper and the previous one is that in a later part of this paper it is assumed that internal events of a composite system are not visible to any users. In particular, the low-level communications inputs and outputs of the components are assumed to be unobservable in the hooked-up system. We feel that this is a questionable assumption. It would require that a distributed system protect its ‘‘internal’’ low-level signals at least as well as they would a high-level signal. This is a strong requirement that is motivated by the desire to protect high outputs.

Finally, the authors require that all channels of a component be classed either as *user* or *system* channels, which we will call interconnection channels to agree with our previous use of the term. Interconnection channels may be hooked together, while user channels are guaranteed never to be. This requirement is also attributable to the protection of high outputs, as will be seen in the next section.

5.4.4.1.1. Non-Deducible Output Security

Guttman and Nadel require on one hand that high-level outputs not be deducible, but they also wish to allow write-ups in the sense that high-level outputs may depend on low-level inputs. To accomplish this, they interpret the write-up as permitting high-level outputs to depend on low-level *user* inputs, but not on low-level communications inputs.

In the notation used in the discussion of the previous paper, the resulting policy ND can be described as:

For every pair of valid traces $a, p \in T$, if event sequence $s \in E^*$ satisfies

$$s \upharpoonright L = a \upharpoonright L$$

$$s \upharpoonright (H \cup (L \cap U \cap I) = p \upharpoonright (H \cup (L \cap U \cap I)$$

then s is a trace.

This is a non-deducibility definition that states that it must be possible for every observable sequence of low-level behaviour $(a \upharpoonright L)$ to coexist with every compatible subsequence that should be protected. In fact, in contrast to a conventional non-deducibility definition that only requires the existence of a single such trace, this definition mandates that every s containing the necessary low and protected subsequences must be a valid trace.

A short example will help clarify this. Let a typical system execution be $a = \langle lui_1, hi_1, lci_1, lo_1, hi_2, ho_1 \rangle$ where lui are low user inputs, lci low communications inputs, hi high inputs and ho high outputs. A low-level observer sees $a \upharpoonright L = \langle lui_1, lci_1, lo_1 \rangle$.

Next, choose any other system execution p that contains the same low user inputs, say $p = \langle hi_3, lci_2, lui_1, ho_2, ho_3 \rangle$. It must have the same low user inputs in order to be compatible with the first trace, but it may differ in all other ways. Now extract from this the combination of high signals (that are supposed to be secret) and low user inputs. We get $p \upharpoonright H \cup (L \cap U \cap I) = \langle hi_3, lui_1, ho_2, ho_3 \rangle$. Finally, merge this subsequence with the previous one, in any desired order. Let us choose $s = \langle hi_3, lui_1, ho_2, lci_1, ho_3, lo_1 \rangle$. If the system has the ND property, this

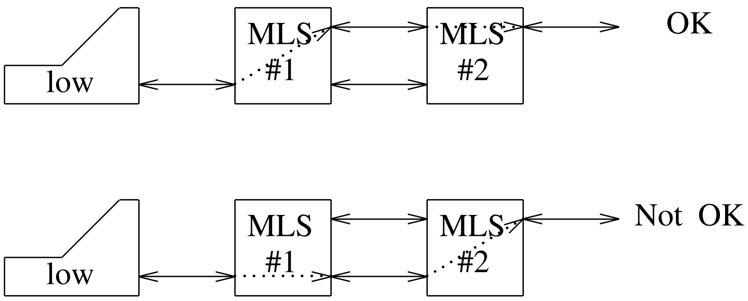


Figure 5.2: Write-Ups Permitted by ND Policy.

sequence must be a valid execution. Therefore, a low user observing $a|L$ cannot tell if he is witnessing the execution a or the execution s. Since the high-level signals of s were selected arbitrarily, except that they had to be compatible with the low user inputs of a, it is clear that the observation of $a|L$ gives no new information about high inputs or outputs. Since the merging was performed arbitrarily, the low observations are also compatible with all interleavings and so give no information about which interleaving occurred.

Inclusion of the low user inputs in both constraints has the effect that the high execution is permitted to depend on the low user inputs. However, they may not depend on low system inputs. This interpretation is validated by the authors on page 48, where they state that their intention was to ''leave high outputs unaffected by low system inputs''.

This latter requirement is very restrictive, and would probably be considered unworkable in many environments. One consequence is that, in a composite system, any write-up must occur in the component to which the low user is directly connected (see figure 5.2). It would also apparently prevent a high level process from reading a low-level file unless that file was known to have been created using inputs supplied by a directly connected user.

5.4.5. Proof of Hook-up

Despite our various objections to the choices made in defining non-disclosure in this paper, the hook-up proof remains of interest because it is constructed along very different lines from those of other papers we have seen. The proof given in the paper is a somewhat complicated proof by induction. The following proof is grown from the same germ but is less complicated and, we believe, simpler to understand.

Theorem

Suppose that for each k, $M_k = \langle E_k, I_k, O_k, T_k \rangle$ is a machine satisfying ND. Let $M = \langle E, I, O, T \rangle$ be the hook-up of the M_k by the permissible hook-up function h. Then M satisfies ND.

To prove this, we must show that if $a, p \in T$ and $s \in E^*$ such that s satisfies the ND constraints, then s is a trace of M. By the definition of composition, $s \in T$ if and only if $s|E_k \in T_k$ for all k.

To begin, we note that for any $x \in E^*$ we have

$$(x|E_k)|L = x|(E_k \cap L) = (x|L)|E_k$$

and, if we apply this specifically to $s|E_k$, we find

$$(s|E_k)|L = (s|L)|E_k = (a|L)|E_k = (a|E_k)|L \qquad (1)$$

We also note that

$$(x \restriction E_k) \restriction H \cup (L \cap U \cap I) = (x \restriction H \cup (L \cap U \cap I)) \restriction E_k$$

and, applying this to $s \restriction E_k$,

$$(s \restriction E_k) \restriction H \cup (L \cap U \cap I) \ = \ (s \restriction H \cup (L \cap U \cap I)) \restriction E_k$$

$$= \ (p \restriction H \cup (L \cap U \cap I)) \restriction E_k \ = \ (p \restriction E_k) \restriction H \cup (L \cap U \cap I) \qquad (2)$$

The combination of equations (1) and (2), together with the knowledge that machine M_k satisfies ND, implies that $s \restriction E_k \in T_k$ as desired.

5.4.5.1.1. Extension to Other Properties

The approach to composability expressed in this proof is very different from that in McCullough's and Johnson and Thayer's work. Their compositions operated by iteratively knitting together sequences representing executions of the components, and using their security properties at each step to determine the perturbations that may result. This paper, on the other hand, uses an algebraic approach that proceeds in the opposite direction. It decomposes an alleged composite execution sequence to show that it its parts are legal.

This approach can be seen to be applicable to a wide class of problems. Any property Ω defined using functions A and B on event sequences, such that if

$$A(s) \ = \ A(a)$$

$$B(s) \ = \ B(p)$$

then s satisfies Ω, can be demonstrated in this way as long as the functions commute with the $\restriction E_k$ operation that restricts a sequence to a particular component machine.

This method can not be used to show composability of the Johnson and Thayer property, because it fails to fit this pattern on two accounts. First, they call a system secure if there exists *at least one* composite execution satisfying a set of constraints. The pattern above will only work if the constraints are to be met by *all* candidate executions, as is the case in the current paper.

Secondly, the functions defined for the Johnson and Thayer work do not commute with the machine restriction operation. This is because, in that discussion, the notions of system and communications inputs/outputs were defined differently when applied to the components than when applied to their composition. In contrast, this paper classes signals uniquely as user or communications signals no matter from what level the system is being regarded.

Although we do not find the definition of security appearing in this paper to be suited to our needs, its approach to composition may prove to be useful. It is a matter for future work to determine what other properties can be profitably defined according to this pattern.

5.5. MARCUS AND REDMOND

In their paper "A Model-Theoretic Approach to Specifying, Verifying, and Hooking Up Security Policies"[Marcus88], Marcus and Redmond propose a solution to the composition problem for systems characterized by a "read" security policy. The "read" concept[Marcus86] of information flow is informally described as follows:

> There is an "adversary" observing how a given "open" variable *v* "behaves" during the computation of some process. The adversary knows some facts about the process, the "public knowledge" K. He is trying to deduce additional information about some "protected" variable *x*, based on *v*'s behaviour and the public knowledge. If he can, we say that *v* "reads" *x* with respect to K. Some behaviour of *x*, a priori consistent with K, is ruled out by the observed behaviour of *v*.

The above informal description can be formulated precisely in terms of models of computations, or "traces". The model can be generalized to encompass other possible security specifications. One generalization given is the addition of two new kinds of variables:

- the common variables: *u* (common to both the "adversary" and the "protector")
- the extra variable: *y* (all other variables in K)

These models lead to precise definitions of the conditions under which the protected variables are or are not "read".

Marcus and Redmond then define the Hook-up problem in terms of their formulation, leading to a set of rules for hooking up "read" secure components to yield a secure composite machine. The hook-up rules are:

- identify and equate variables in the different machines (models) which represent the channels to be hooked together
- take the conjunction of the altered descriptions of the different machines to obtain the description of the composite machine
- forget (possibly) variables representing channels which after they are linked become unobservable
- common variables and extra variables may not be forgotten, nor may the latter be hooked-up.

The paper examines two 'classic' hook-up problems. The Guttman and Nadel system consists of two processes, A and B. A has one low security input and one low security output which is non-deterministically based on the input. B has one low security input which is copied to a low-security output and a high security output. They are secure according to the Guttman and Nadel definition. The composite formed by connecting the output of A to the input of B is insecure in the Guttman and Nadel view, and therefore that method of hook-up is made illegal. According to the "read" concept, B is already insecure, therefore hook-up is not of interest.

The second hook-up example is that of [McCullough88c], which we have discussed earlier. Again, the Marcus and Redmond "read" concept yields the result that one of McCullough's components is insecure, so again, hook-up is not of concern. (An alternate formulation, in which the components are secure, obtains that the proposed hook-up violates the Marcus and Redmond hook-up rules, and thus is illegal.)

Unfortunately, we find in this work nothing of value to us in our approach to composability. We find the concept of "read" security to be excessive in the sense that it rejects systems that, in current practical terms, are secure. As an example of this difficulty, consider the following simple mathematical example of public knowledge. Suppose it is public knowledge that

$$\text{if } v = 5 \quad \text{then} \quad x \neq 5 \quad \text{with probability} \quad .001 \ .$$

According to Marcus and Redmond, the observation of v, when it is 5, reveals new information about the protected variable x. Therefore the system is not secure. We find this to be too strong a result to be of practical use.

6 INTEGRITY AND DENIAL OF SERVICE

Does one's integrity ever lie in what he is not able to do?
Flannery O'Connor,
Wise Blood (foreword),
1952.

In silence man can most readily preserve his integrity.
Meister Eckhart,
Directions for the Contemplative Life,
c. 1320.

6.1. INTRODUCTION

Most, if not all, of the existing published work in composition of security properties focuses on the relatively well-understood and well-formalized domain of mandatory non-disclosure. Perhaps this can be partially explained as a simple continuation of the security community's habit of emphasizing non-disclosure, but this is likely not the entire story. The pre-existing body of formal models and definitions of non-disclosure supplied the proper foundation on which theories of composition could be built. Several of these were examined in detail earlier. The number and variety of approaches is surprising, given the infancy of the subject.

Formal definitions and models of other aspects of security are either rudimentary and not widely accepted, or are altogether lacking. Again, this is partly a result of an historical preoccupation, but is also reflective of real difficulties inherent in the formal treatment of these other security concerns.

Informal treatments are somewhat easier to come by. One reasonable, though certainly not universally acceptable, intuitive understanding of immunity from denial of service attacks can be phrased as ''the system will continue to function despite the actions of unauthorized individuals.'' By the phrase ''continue to function'', we mean that the system will function in as correctly and as timely a manner in the presence of interfering activity as without. In this informal setting, use of the words ''as correctly'' enables us to rather easily decouple denial of service from other system faults that might cause incorrect operation.

For example, if an untrusted application program contains a bug, then the system when running that program will not meet its operational objectives and so cannot be said to be operating correctly. However, in the sense that this is a ''legitimate'' bug (an awkward phrase, but the only one that comes to mind), this is not a denial of service problem as defined above, and perhaps not really a security problem of any stripe.

But, if the application did not have the bug yesterday and the system permitted its introduction by someone not authorized to modify production software, then this might be classed as a denial of service. It could, of course, also be called other things, but if we accept the view that the production software is a system resource that has been rendered unusable, the denial of service label does seem to fit.

This distinction can be made informally with relative ease, but the discipline of formal specification does not permit such shortcuts. A precise definition of ''denial of service'' is difficult to achieve without a similarly meticulous definition of exactly what ''service'' is.

This difficulty is shared by other security attributes, and is one of the fundamental ways in which they differ from non-disclosure. Nondisclosure is entirely a *safety* property — it makes no demands on what the system must do, but does does say what it must not do. Furthermore, nondisclosure is readily defined and enforced at the application boundary. The application's internal function is unimportant, as is the intended meaning of its inputs and outputs, because nondisclosure can be defined as the lack of correlation between uninterpreted message streams.

This definition process can also be carried out at the subsystem level, where we again find that the nondisclosure properties of a subsystem can be defined in terms of the patterns of its inputs and outputs without reference to their meaning. Knowledge of the component function is eventually necessary in order to represent that particular component in the model, otherwise we cannot characterize it well enough to measure it against the nondisclosure property definition, but this knowledge is not necessary when dealing with the property in the abstract.

This characteristic greatly simplifies the nondisclosure composition problem. The security of the composite system can be defined without regard to its intended operation, and without regard to the interpretation of the signals exchanged by the components. Thus, composable non-disclosure can be defined without requiring that the components work together in a correct way, or even in a meaningful way.

The same cannot be said of other security properties. Integrity, denial of service protection, audit, authentication, etc. all imply certain correct function of a system, and cannot be suitably defined for a ''black-box'' system with uninterpreted input and output ports. For this reason, techniques developed in modelling nondisclosure and in investigating its composability are not necessarily directly transferrable to other security areas.

In this report, we will consider several types of models and specification systems proposed for various security properties. Each model will be evaluated with respect to its composability properties, and also for any characteristics that might induce a meaningful order among systems evaluated against the model. Our objective here is not to exhaustively enumerate all possible models, but rather to obtain a cross section of security modelling techniques that we hope will be representative of the breadth of models that may be proposed elsewhere in the future.

6.2. THE YU-GLIGOR DENIAL OF SERVICE MODEL

Yu and Gligor [Yu88] describe a model which, in their words, provides

> a formal specification and verification method for the prevention of denial of service in absence of failures and of integrity violations.

This definition is that promulgated by Gligor, and is almost the antithesis of the informal definition attempted in the introduction to this chapter. Its arena of specification is the managers of shared resources, since denial of service problems are all related to the sharing of resources. The threats considered are the withholding of service and the provision of incorrect service. The model assumes that users are authorized and are not malicious, and so closely resembles and borrows extensively from the theory of concurrent program verification.

At the foundation of the method lies a need to make and prove statements of the form ‘‘all users requests eventually progress’’. The formal language chosen for this is *temporal logic*.

6.2.1. Temporal Logic

Temporal logic is an extension of first-order logic that permits the truth value of logical statements to vary with time. To allow quantification over time, it introduces the modal operators $\Box$ and $\Diamond$, with the semantics

$$\Box\ P \equiv \text{``P is true now and will remain true''}$$

$$\Diamond\ P \equiv \text{``P is true now or will become true sometime''}$$

These operators, pronounced *henceforth* and *eventually* ($\Box$ and $\Diamond$, respectively) can be combined in several useful ways:

$$\Box \Diamond P \equiv \text{``P will be true infinitely often''}$$

i.e. P may be false or become false, but each time it does it will later become true again.

$$\Diamond \Box\ P \equiv \text{``P will eventually become true and remain so''}$$

$$P \leadsto Q \equiv \Box (P \Rightarrow \Diamond Q) \equiv \text{``P will always eventually lead to Q''}$$

as well as the fundamental

$$P \text{ UNTIL } Q \equiv \text{``P is true at least until Q first becomes true, if it ever does''}$$

With these tools, the resource manager specifier defines a set of temporal formulae that represent assertions about the module's operation. These assertions are grouped into two sets of internal correctness constraints and three sets of policy specifications.

The correctness constraint groups are the *resource constraints*, which are invariants about the managed resources (eg. the free plus in-use counts always add up to the total number of resource units, or $\Box (free + inuse = total)$), and the *concurrency constraints*, which express any mutual exclusion requirement that must be met within the manager (eg. $\Box (\#Active = 1 \text{ or } \#Active = 0)$). These, together with the specifications of individual services contained in the bodies of the service routines, are used to prove that if the resource manager provides service at all, then that service is provided correctly.

The question of whether service is ever provided is the subject of the three policy assertion groups. They consist of the *fairness policy* specification, the *simultaneity policy* specification, and the *user agreement* description.

The fairness policy describes the fairness of scheduling among active requests that are competing for service and all eligible for satisfaction. In the examples given in the paper, they seem to consistently be of the form

> if a process invoking a service has infinitely many opportunities to progress, the service will eventually be provided

which, for an allocation service, might be written

$$at(Allocate) \text{ and } \Box \Diamond (free \geq \#requested) \leadsto after(Allocate)$$

The simultaneity policy states that, if the users are well-behaved, then all the conditions for progress of a blocked service request will be simultaneously satisfied infinitely often, and so will afford the ''infinitely many opportunities'' required by the fairness policy.

The user agreement section contains the definition of ''well-behaved''. It describes a protocol for using the shared resource. All users are required and trusted to follow the protocol. It establishes the necessary conditions for the operation of the simultaneity policy. It may contain assertions that resources will be freed only after they are allocated, or that all resources allocated will eventually be freed. They may also express policy that exists to avoid deadlock between users of multiple resources, such as Havender's policy [Havender68] of always allocating several resources in a fixed order.

It would appear that some of these user agreements might be enforced by the system (the allocate before free requirement, for example), but at least some are clearly not within the domain of a particular resource manager, and so in the specification the user agreements are formally separated from the service specification. Also, some of the methods a system might use to enforce these agreements, such as detecting incipient deadlock and aborting one user, are themselves disruptions of service and so could not by themselves be considered a valid response to the denial of service threat.

6.2.2. Type of Model

This is a very young model and assuredly does not represent in detail what we may expect as future denial of service models. Although it claims to be a verification platform, the policy assertions it contains are intimidating statements to try to verify and appear, in fact, to be treated as axioms in the verification. Such verification as is done is handled informally and, at times, incorrectly.

A sample correctness assertion given in the paper

$$own\,[id\,]{=}M \text{ and } \Diamond after\,(Allocate\,(n\,)) \Rightarrow \Diamond(own\,[id\,]{=}M{+}n\,)$$

is not the assertion that really should be made for the module under discussion. The $own\,[id\,]{=}M$ condition should be specified to hold at the entry to the $Allocate\,(n\,)$ service, but in the assertion as given is not tied to any specific time. Also, the updated value of $own\,[id\,]$ is permitted to hold at any time in the future, rather than at the specific time when the $Allocate\,(n\,)$ service is complete. A better expression would be

$$own\,[id\,]{=}M \text{ and } at\,(Allocate\,(n\,)) \Rightarrow \Diamond(after\,(Allocate\,(n\,) \Rightarrow (own\,[id\,]{=}M{+}n\,))$$

The first problem, at least, would have been discovered if a formal verification had been performed, since the assertion in its original form would not be provable.

These shortcomings are pointed out by way of illustrating that this formalism has not yet matured. Still, it is of interest as a representative of the current art in this area, and as a rudimentary example of a temporal logic specification system of a kind that might be encountered in more sophisticated models to come.

6.2.3. Measure

The type of properties eventually proven about the behaviour of a single shared resource are progress assertions, such as

$$at\,(Allocate\,(n\,)) \leadsto after\,(Allocate\,(n\,))$$

eg. all invocations of the *Allocate* service eventually return, and correctness assertions as discussed in the previous section. The following are obvious ways in which individual shared resources meeting these specifications might be ranked:

1. by strength of the progress assertions;
2. by service-specific ordering of the correctness assertions; and
3. by the inverse number and complexity of user agreement restrictions that the users of the resource are trusted to obey.

Treating each point in turn:

Progress Assertions

The current notation does not seem to permit different strength progress assertions to be made, but it might be reasonable to expect this ability as a future development. Currently, the progress assertions simply state that invocations complete in a finite but unbounded time. Stronger assertions one might like to prove and which, *ceteris paribus*, would result in a higher measure of the service guarantee, might take the form of strong real-time guarantees (bounded response times) or perhaps guarantees about the relative order of satisfaction of competing requests. The latter might be as simple as the FIFO discipline described by

$$(user\ 1.at\ (Allocate\ (n))\ \text{and}\ \Diamond user\ 2.at\ (Allocate\ (n)))$$
$$\leadsto$$
$$user\ 1.after\ (Allocate\ (n))\ \text{and}\ \Diamond user\ 2.after\ (Allocate\ (n))$$

which states that if user1 and then user2 make the same request, user1 will be satisfied no later than user2.

Given two alternative sets of progress assertions, if one implies the other then the first is ''stronger'' in the mathematical sense. From a practical point of view, it is possible that the extra constraints of the stronger assertion represent undesirable behaviour. For example, one system may only guarantee eventual completion while a second system additionally specifies that all requests will take at least five seconds. In such cases, it should be noted that the more weakly constrained system was also capable of this behaviour, so we might be justified in saying that the more constrained system has the better guarantee if for no other reason than it gives a better idea of how bad things can be. In the usual case, where neither system's assertions imply the other's, no simple guideline is obvious and the relative rating of the two systems must be determined by a knowledgeable individual.

Correctness Assertions

These also afford no obvious scale for comparison, but it is probably less of an issue here than in the previous case. This is because in the normal situation, if correctness assertions of two shared services differ significantly, then they are offering different services and it is meaningless to attempt to compare them. We therefore expect that these assertions will not form a basis for guarantee comparison.

User Agreements

These represent how much the resource users must be trusted in order that the performance assertions provided will be valid. Therefore, weaker user agreements indicate a lesser threat from broken or malicious users, and so a higher guarantee rating should be expected.

This question is clouded by interactions with other security mechanisms. A resource with strong user agreements (i.e. requiring much trust) but which is protected by stringent access controls may be preferable to a wide-open resource with weaker agreements. For this reason, a realistic measure of guarantee cannot be made on the basis of this model alone.

The factors that should be considered in such a determination are the complexity of the user agreements, the strength of protective access controls, and the extent of damage done to the correctness and progress assertions if the user agreements are not obeyed.

6.2.4. Composition

Composition may take the form of the hook-up of subsystems containing shared resources. Four classes of hook-up effects are easily recognized:

1. effects due to interactions between parts of a distributed shared resource manager.
2. conflicts between resource managers that no longer have exclusive control over a resource shared between components.
3. conflicts between users of a resource which previously had no sharing management, or which had implicit management because it had only a single ''user''.
4. effects due to user's inability to simultaneously maintain user agreements with respect to shared resources in multiple components.

The first three should be forseeable and avoidable. Each of them depends on the hook-up facilities offered by the components and should be predictable from the study of each component in isolation. If a component has ''ports'' for hook-up, its designers should be able to determine what resources are potentially sharable through those ports. The most error-prone part of this exercise is identifying newly-shared resources that were previously single-use, since there will be no pre-existing sharing to draw the designer or certifier's attention to the resource.

The fourth part is the most difficult. Within a single component, user agreements are used to avoid deadlock, but they are sufficient only if they encompass all sources of service delay or denial. Introducing new components with new shared resources will introduce new ways for subjects to deadlock one another.

A particularly interesting case occurs when the interconnection itself is a limited resource, shared naturally by the two components and possibly also shared in a subsidiary way by all subjects active on each of the two components. If a hook-up can be made in which a response message from each component solicits a new response from the other, the channel may be saturated by this activity preventing any useful communication. Analysis of the components in isolation may not uncover this problem, particularly if during that analysis the interconnection is deemed to terminate in a user, because from the viewpoint of the component its ‘‘user’’ is indeed getting service.

6.3. THE CLARK-WILSON INTEGRITY MODEL

This is a model of a different sort that was first presented in [Clark87] and which currently enjoys a great deal of popularity. It may fairly be characterized as an elaboration of the trusted pipeline concept due to Boebert and Kain [Boebert85].

The Clark-Wilson ‘‘model’’ consists of a set of procedures and mechanisms intended to achieve a form of integrity control required by commercial data processing users. The class of threats addressed is the incorrect modification of data through any of error, malice, or fraud.

The controls described in are intended to constrain the ways in which certain critical data items are altered. Identification of the target **Constrained Data Items** (CDIs) is the first step in implementing this system. Next comes the definition of a set of **Integrity Validation Procedures** (IVPs). These are audit procedures that check whether the CDIs are in a valid state at the time the IVPs are invoked. In general, this means checking internal and inter-CDI consistency as well as verifying that the CDIs accurately represent those features of the external world they are intended to track.

The third step is the validation of **Transformation Procedures** (TPs), the actual transaction procedures that will be applied by users to the CDIs during normal operation. Their validation involves gaining assurance that they will never transform a valid CDI state (eg. one which satisfies the IVPs) into an invalid one.

Once these preliminaries are completed, operation of the system is governed by a set of nine rules. Five of these relate to certification and the other four to enforcement. Briefly, these are:

C1: All IVPs must ensure that all CDIs are in a valid state.

C2: Each TP must be certified to transform valid CDI states only into valid CDI states.

E1: The system must maintain, for each TP, a certification list of CDIs with respect to which the TP has been validated. It must further ensure that CDIs are manipulated only by TPs certified for use with that CDI.

E2: The system must maintain a user authorization list of triples of the form (UserID, TP, list-of-CDIs) and must prevent TP executions not described by any triple.

C3: The triples list of E2 must be certified to meet installation-specific separation of duty requirements.

E3: Each user must be identified before executing any TP.

C4: Every TP must be certified to write to an append-only CDI (log) thereby providing an audit trail.

C5: TPs that accept non-CDI input data must be certified to transform valid CDI states into valid CDI states for any value of the non-CDI input.

E4: An authorization list may only be changed by a certifying authority. That authority must not have any execute rights with respect to the entities governed by the list.

6.3.1. Type of Model

Although Clark and Wilson describe this as a formal model, it is not a model in a sense that would make it very useful for our purposes. It is not a mathematical analogue of a system that we can manipulate to see how it reacts to varying circumstances. Neither is there a formal definition of the integrity goal, and no proof that the procedures and mechanisms described will suffice to meet its informally defined goals of data consistency and dependability in the absence of collusion.

Although other researchers have produced models that answer some of these criticisms, it remains that the Clark-Wilson approach and the definition of integrity that it implies have received the most positive reaction from the security community. Even so, there is still not universal agreement on what ‘‘integrity’’ should mean or what ‘‘integrity controls’’ should be controlling. Given the controversy about definitions, even informal ones, it is not surprising that a model suited to our present needs has not yet been developed.

6.3.2. Measure

Clark and Wilson address the issue of evaluating integrity features in [Clark88]. They identify three levels that are intended to complement the TCSEC certification divisions A, B, and C.

	Division		
Feature	A	B	C
Prevention of Change	R/D	R/D	O/D
Data Labels and Logs			
——change log	R/M	R/M	R/D
——change log with attribution	R/M	R/D	O/D
——IVP execution log	R/M	R/M	O/D
——domain logs for data	O/D	O/D	--
Access Control Triple	R/M	R/M	R/D
Application Program Change Control	R/M	R/M	R/D
Uncircumventable User Authentication	R/-	R/-	O/-
Controls on Privileged Users	R/M	R/M	O/D
Dynamic Tracking of Separation of Duties	O/D	O/D	--

M = Mandatory D = Discretionary R = Required O = Optional

The presence or absence of these features can be used in this way to define at least three distinct grades of integrity guarantee. Jueneman [Jueneman89] builds on the Clark-Wilson foundation, proposing a richer and more complete set of integrity features for which a similar but more elaborate table might be constructed. Examples of the extra features he adds are a *trusted date and time facility* , so that TPs that should only be run at certain times (eg. the end of a month) can enforce such restrictions reliably, and *trusted sequencing* facilities to ensure that TPs are executed in a specified order.

The strength of individual mechanisms is also a plausible basis for comparing integrity guarantees. This is already present to some extent in the table, eg. the uncircumventable user authentication required at higher levels, possibly implemented by an intelligent card, presumably replaces a weaker mechanism such as password-based authentication.

6.3.3. Composition

Since the Clark-Wilson system is a description of integrity mechanisms rather than a representation of their operation, it cannot be used to investigate dynamic interactions between interconnected systems in the same way that trace-based non-disclosure models can. Here, all that is visible are the rules themselves, so our analysis is limited to the support for uniform interpretation and enforcement of these rules across the composite system.

This does not necessarily mean that the same full set of rules must be enforced in each component. What is required is that the controls contained in one component not be evaded by following an alternate access path through another component.

It is notable that the integrity guarantee measure of a composite system may routinely be higher than that of any of its components. Provision of high-quality integrity controls involves implementing several loosely-coupled mechanisms, and it is quite possible that the components may implement complementary parts of the whole. For example, a front-end unit may contain a sophisticated user authentication mechanism, while the transaction processing engine may have no authentication capability at all.

This possibility creates new problems for composition analysis. The composite system satisfies the integrity requirements only if all user access is constrained to occur through the front end subsystem. Unlike compositional properties we have seen before, this property is not associated with any of the components. It relates solely to how they are interconnected, and to how they are ‘‘connected’’ to the community of users.

Even if we restrict ourselves to the composition of units that are relatively self-contained integrity-preserving subsystems, there are no obvious composition rules that present themselves. Consider a system whose components with varying levels of integrity guarantee are interconnected, but where a policy is instituted that all CDIs must reside on and be modified through the component with the highest guarantee. Other components may contain read-only copies of the CDIs, but authoritative updates happen only on the one subsystem. It should be possible for our composition rules to recognize the contribution of this policy to the strength of the overall system integrity controls.

An analogous situation in non-disclosure terms would be a single multilevel system connected to a number of single-level machines. Because non-disclosure deals in sensitivities, and the interconnections can be modelled as single-level links, a policy of restricting multilevel activity to a single machine is directly visible in the model. Looking at it another way, the components with only single-level ports can be trivially certified as being hook-up secure. Their single-level operation, though it is actually a result of an interconnection policy and not (necessarily) an innate property of any of those components, is directly visible in the interconnection model.

Integrity composition needs its own interconnection model where the policies governing the interconnection are similarly visible. It will probably not be the same model used for non-disclosure analysis, because the links between components will have to be characterized in terms that are meaningful to the integrity control mechanisms.

Another approach would be to concentrate on one or more of the more fully developed models that attempt to implement Clark-Wilson style integrity controls. Some of these use re-interpretation and some extra administrative procedures to graft Clark-Wilson policies onto older security models. Lipner [Lipner82] and Lee [Lee88] both use Biba-style integrity levels and categories as the basic mechanism to implement commercial integrity, though only the latter is recent enough to have been influenced by Clark-Wilson. These have the disadvantage that the model is not really constructed with these policies in mind, so the results of any analysis might be specific to that method of implementing the policies rather than generally true of the policies themselves.

Also, some of these implementations only approximate the Clark-Wilson rules. Lee's system, for example, does not provide a true $(user, TP, list-of-CIDs)$ triple, but attempts to synthesize it from a combination of user/TP, user/CID, and TP/CID access controls. The result is somewhat clumsy, and might only deliver an approximation to the Clark-Wilson triple.

Other formal integrity models are epitomized by Pittelli [Pittelli88] who uses a non-interference approach to formalize a limited kind of separation of duty. His model splits commands into command-authorization pairs, and requires through a non-interference style of specification that all commands be matched up with appropriate authorizations. This is a promising modelling approach, but captures only a small part of the Clark-Wilson policy set.

6.4. NONDISCLOSURE MODELS AND META-MODELS

Earlier we discussed the models of McCullough [McCullough87] [McCullough88c], Johnson and Thayer [Johnson88], Guttman and Nadel [Guttman88], and Marcus and Redmond [Marcus88]. These are all well-developed formal models. All were designed with composability in mind, and all can be used to analyse dynamic interactions in composed systems.

There also exist a number of other models, not specifically designed for composability assessment, but which are interesting because of other properties they possess. This section will survey some of these unusual models.

6.4.1. McLean's Security Algebra

John McLean [McLean88] describes a framework for developing and comparing specific security models. These are basically access-control models in the style of Bell and LaPadula, with the added ability to control the authorization to change subject and object security levels. Models expressible within the framework are differentiated by their restrictions on this level-changing ability.

McLean also develops a variant framework that allows the expression of N-person rules, an important constituent of integrity policy. Both frameworks exhibit a natural ordering of the models they can express, and both contain natural and simple operators for formally composing policies.

The basic access-control model is unsurprising. A system is modelled as a state machine. The notation used is:

S	subjects
O	objects
L	levels
A	access modes, eg. {*read, write, execute* }
v	$= (b, f)$ is a system state where
b	$\subset \mathbf{S}\times\mathbf{O}\times\mathbf{A}$ is the current access set
f	$= (f_s, f_o)$ are the security level mapping functions for subjects and objects

State changes in the model consist of altering the set of granted accesses, changing the levels of some elements, or both. Models are distinguished by one other characteristic, the pair

C	$= (c_s, c_o)$ where
c_s	is a function from $\mathbf{S} \rightarrow \mathbf{S}^*$
c_o	is a function from $\mathbf{O} \rightarrow \mathbf{S}^*$

These two functions specify which subjects are permitted to change the levels of each subject or object, respectively. Note that **C** is not part of the state, so these authorizations are not alterable by normal operation of the model.

Simple security and $*$-security properties ar defined in the usual manner. The author explicitly disavows any statement that these properties are in themselves a definition of security. Instead, models within the framework are permitted to have different security policies as long as they include the SS and $*$ properties.

Frameworks are formally distinguished as sets of models defined on a given **S**, **O**, **L**, and **A** sets. Models within a framework may differ in their **C** components, and this difference is the basis of an ordering defined on the models.

McLean reasons that there exists an intuitively obvious ordering of models within a framework. Models in which all subjects can change the levels of all subjects and objects (i.e. $c_s(s)$=**S** and $c_o(o)$=**S**) are at one end of the scale. Models in which no subjects are given any powers of level alteration (i.e. $c_s(s)=\emptyset$ and $c_o(o)=\emptyset$) are at the other end. He completes the definition of a partial order

$$
\begin{aligned}
&M_1 \le M_2 \\
&\text{for } \; M_1 = \{\mathbf{S},\mathbf{O},\mathbf{L},\mathbf{A},\mathbf{C}_1\} \\
&\qquad M_2 = \{\mathbf{S},\mathbf{O},\mathbf{L},\mathbf{A},\mathbf{C}_2\} \\
&\text{iff } \; \mathit{foralls} \;\; c_{1s}(s) \subseteq c_{2s}(s) \\
&\text{and } \; \mathit{forallo} \;\; c_{1o}(o) \subseteq c_{2o}(o)
\end{aligned}
$$

Actually, $c_{1s}(s)$ is required to be isomorphic to a subset of $c_{2s}(s)$ etc. This detail is omitted for the sake of clarity.

Models lower in this ordering are more restricted, in that they permit fewer subjects th privilege of altering subject or object security levels.

The partial order defined here is essentially that of set inclusion. Set inclusion defines a Boolean algebra with intersection, union, and set complement as its greatest lower bound ($\wedge$), least upper bound ($\vee$), and complement operators, respectively. Therefore, this characterization of nondisclosure models leads to a Boolean algebra of nondisclosure policies.

6.4.1.1. Application to Guarantee and Composition

A partial ordering of models within a framework is similar to the guarantee comparison principles suggested for other models earlier in this report.

Reasonable guarantee measurement schemes will have the transitivity property, such that if guarantees $G_A \ge G_B$ and $G_B \ge G_C$ then $G_A \ge G_C$. It would be inconsistent with our intuitive notion of measurement for this not to hold. On the other hand, it would be unreasonable to require that this be a total order.

Composition of guarantee is where the properties of the Boolean algebra become useful. If two systems from the same framework are combined so that their permissions add, we will have

$$s_1 \in c_s(s_2) \;\text{ if }\; s_1 \in c_{1s}(s_2) \;\mathit{or}\; c_{2s}(s_2)$$

and the resulting system is the least upper bound of the components. If, instead, the combination accumulates the restrictions of the subsystems, the result is the greatest lower bound of the component systems. The Boolean algebra operators are powerful tools for formally combining and describing complex policies. Change of level authorization sets are only one basis for defining a guarantee measure, and perhaps not a very useful one. But, this example does show that models can be well suited to formal manipulation and combination.

6.4.1.2. N-Person Extension

McLean then modifies the model to allow for the expression of N-person rules. To achieve this, functions and state variables in the original model that are defined for single subjects are replaced by corresponding ones defined on sets of subjects. The interpretation of a non-singular set in the access matrix or as a constituent of a level-change authorization function

$$\{s_1, s_2, s_3\} \in c_o(o_\alpha)$$

is that the subjects listed have joint access or authority to perform the indicated operation.

After some tuning to eliminate nonsensical policies, a framework results that is a distributive lattice of policies rather than a Boolean algebra. Elements of distributive lattices are not guaranteed to have a unique complement, but the greatest lower bound and least upper bound operators still exist. These therefore have almost as great an appeal as the Boolean algebras.

6.4.2. Jacob's General Security Specification

Jacob [Jacob88] introduces a model which he promotes as suitable for the representation of many security properties. He traces its provenance to Hoare's work on the trace semantics of communicating sequential processes (CSPs) [Hoare85]. This is not unique, since there exist many trace-based modelling schemes and they owe a debt to Hoare whether they recognize it or not. This model differs from those most obviously in its notation, but more importantly in the ambitious range of applications that Jacob attempts.

System components (''processes'') are characterized by the prefix-closed sequences of external events they may engage. Users are identified by the subset of events they can observe. Equivalently, ports on a system component can be defined in the same way. The notation

$$\text{infer } S\ B\ l$$

denotes the set of possible executions of system S that are consistent with user B's observation of the event sequence l.

This is a departure from the non-deducibility work of Sutherland [Sutherland86], whose concern right from the beginning was with the inferences one user can make about another's behaviour. Jacob takes one step back, and initially concentrates on what an observer can deduce about the operation of the entire system. From this, Sutherland's concerns can be addressed through the projection (infer $S\ B\ l)\uparrow C$ to yield the behaviour of user C inferrable by B from his observation l.

Jacob defines a mandatory confidentiality policy in terms of these inference functions. The expression of the policy is Goguen and Meseguer's non-interference property adapted to Jacob's notation. It requires that any observations made by B must be consistent with C having executed no events.

Jacob next attempts to use the same tools to formally define trusted users and integrity. Trusted users (or ''subjects'') are users whose interface to the system gives them the opportunity to violate confidentiality, but which are trusted not to do so. They are incorporated into the model by assuming that the trusted user's behaviour can be (at least partially) characterized by traces, such that it is known that no dangerous traces occur. The user is then connected onto the system to form a larger composite system which can be shown to be nondisclosure-secure.

Integrity is dealt with as the level-based dual of nondisclosure. This is an elementary and restricted interpretation of integrity that has recently fallen out of favour. Instead of showing that a low-clearance user's observations cannot be influenced by a high-clearance user's behaviour, the dual requirement is that a high-integrity user's behaviour cannot be influenced by a low-integrity user. Since this model is closely related to non-disclosure, the same sort of non-interference statements can be used to formalize it.

Inference functions are also shown to be useful in specifying more precise and application-specific security requirements common in real-world situations. It is often necessary that one user be permitted some limited knowledge of another's behaviour, but at the same time be prevented from knowing other aspects of that behaviour. Limited but non-zero permitted knowledge of this type is specified using generalized cousins of the inference function, which Jacob calls *security specifications*.

If f is a security specification, then, like an inference function, it defines a set of system behaviours

$$f\ B\ l \subseteq powerset\ (A^*)$$

that are required to be consistent with the observation of l by B. The difference is that f, unlike infer S, is not a total function. It is defined only for those B and l that are of particular interest to the specifier.

A system S satisfies such a specification iff

$$\begin{aligned} &forall B \in \mathrm{dom}(f):\ \mathrm{dom}(\mathrm{infer}\ \ S\ B) \subseteq \mathrm{dom}(f\ B) \\ &forall l \in \mathrm{dom}(\mathrm{infer}\ \ S\ B):\ (\mathrm{infer}\ \ S\ B\ l) \subseteq (fBl) \end{aligned} \tag{1}$$

Clause (1) requires that, for each view B dealt with in the specification, the observations that may be made of the system are no greater than those listed in the specification. In this way, the specification limits what is directly observable through selected views.

Clause (2) asserts that, for any observation l through view B, the system must have at least as many plausible executions consistent with that observation as the specification does. Inferences based on observations are therefore no more certain, and perhaps more uncertain, in the compliant system than in the specification.

6.4.2.1. Measurement

If a system satisfies a specification, this may be phrased as ‘‘system S is at least as secure as specification f’’, at least insofar as information flow is concerned. Specification satisfaction is an order-like relation between systems and specifications.

Inference functions, which characterize the system and appear in the definition of satisfaction, are really just totally defined specifications. In other words, they are specifications with all of the uninteresting (from a security perspective) detail added in. Therefore, the notion of comparison can be generalized to permit similar strength comparisons of systems against systems, or specifications against other specifications.

System S can be said to be at least as (non-disclosure) secure as G if S allows no more observations than G for all views, and does not permit more precise inferences for all views and observations. Similarly, security specification f is at least as secure as g if f is defined for (i.e., constrains) at least all the views that g constrains, allows no more observations through those views, and does not permit more precise inferences.

This relation constitutes a partial order on the space of inference functions (systems) and specifications. Jacob further shows that, for any given alphabet of events, the partial order defines a complete lattice of specifications/systems.

6.4.2.2. Composition

The treatment of trusted users involved incorporating a trace specification of the trusted user into the model and thereby forming a new, composite model with different security properties. This is an example of composition, though of an unusual kind because the composite system has better security properties than its components. This can happen in this particular instance because the trusted user is known to exercise only a safe subset of its interface functionality, and by occupying one ‘‘port’’ of the central system it prevents other untrusted users from using that port to perform unsafe executions. This is a composition possibility that is not dealt with by the mainstream composition models.

In addition, the lattice of security specifications gives us the kinds of formal tools for combining guarantees that were available in the McLean model.

Despite its pretensions, this model really only treats nondisclosure well. Integrity is handled only in an unimpressive form, and with some strain, by dressing it up as nondisclosure. A more practical integrity model that maintains the lattice properties would be a definite asset for this approach.

6.4.3. Marcus Turing Machine Model

Marcus [Marcus89] proposes the use of the Adversarial Turing Machine formalism as a platform for the specification and analysis of general security properties. This is a derivative of the use of interacting Turing machines in zero-knowledge proof systems [Goldwasser85]. In these systems a definition of nondisclosure is required that is more pragmatic than non-interference or non-deducibility. Eg. observation of a public key may theoretically enable the calculation of the corresponding private key, but that computation is known or believed to be impractically slow. Turing machines are used to define the set of feasible computations, and a system is secure if prohibited information cannot be effectively computed by the observer. Some definitions consider even the case where the observer's computations use algorithms that succeed only probabilistically. Marcus' use of the Turing machine formalism does not appear to be quite this ambitious, although modelling an observer's computational resources might also be a useful technique in large scale security models.

The ATM is a deterministic 2-tape TM that can simultaneously move on both the D_x and D_y tapes. The two tapes can be interpreted as two users or as two ports by which the system interacts with its environment. A user represented by one tape sees the tape contents after each move, and so is aware not only of the data on the tape but the manner in which it is modified. In confidentiality terms, the contents of the two tapes and the sequence of modifications that produced them can be considered information of two security levels. The state of the ATM's finite control is a pair (s_x, s_y) so that we can think of the two-tape ATM as being a pair of Turing machines bolted together, which interact through their shared control.

Nondisclosure is defined similarly to the way it is done in non-deducibility: information does not flow between ''ports'' if all computations on one tape can coexist with each of the computations that can occur on the second tape. That is, an observer watching one tape and who has some knowledge of the construction of the ATM control can make no deductions about what is or isn't occurring on the second tape.

The other security definitions attempted are much more vague. One half of the ATM *writes* on the other half if, eg., the contents of the D_y tape can change without the s_y state component changing. That is, the D_y tape changes without the y ''half'' of the ATM being aware that something has happened. A ''write policy'' is an ATM control that defines which of these writes are permissible. An ATM violates a write policy if the ATM's program allows one tape to write on the other but the policy does not.

Denial of service is defined with respect to a service policy. This is framed as a set of pairs of s_y state components (s_{y1}, s_{y2}) representing a state transition of the y-half of the ATM that must always occur if the ATM is compliant. This means that whenever the ATM enters a state (s_x, s_{y1}) the next state in the computation must be $(s_{x'}, s_{y2})$. Otherwise, the x part of the ATM has denied service to the y part with respect to the stated policy.

Composition is addressed in a somewhat confusing and non-intuitive way. Two ATMs T_1, T_2 are hooked up by, essentially, connecting both their controls to a single pair of D_x and D_y tapes and tape heads. Pairs of x state components and pairs of y state components in the two controls are identified as corresponding pairs of ''hook-up states''.

At any time, only one of the controls is active. Assume that the control for T_1 is running the composite ATM. If it enters a state (s_{x1}, s_{y1}) for which both s_{x1} and s_{y1} are identified as hook-up states, then the ATM may immediately change to the corresponding (s_{x1}, s_{y2}) hook-up states in the T_2 control. This state change activates the T_2 control and renders the T_1 control dormant until another hook-up opportunity arises.

Basically, this means that T_1's program and T_2's program alternately execute on the same pair of tapes, each picking up where the other left off. The hook-up theorem in the paper appears to simply say that if there are no hook-up opportunities (for example, if no hook-up pairs are defined) then the hooked-up system is as secure as its components. This seems obvious, since the hooked-up system must continue executing under the control of whichever program is initially active.

6.4.3.1. Evaluation

Criticism of this model must be tempered by the realization that it was only intended to be a proposal of a modelling approach that seemed promising to its author. We can restrict ourselves to noting that there are no obvious metrics here for ranking guarantees. Also, the integrity and denial of service definitions are too primitive to say anything meaningful about.

The composition rule is well-defined but does not appear to me to capture very well the notion of two ATMs connected together to form a larger system. It seems closer to two ATMs running as users sharing hardware (the tapes) and ''time slicing'', but even this parallel is faulted because when a control resumes it does not necessarily start up in the same state (i.e. point in its program) at which it was suspended. A better intuitive feel for the nature of this hook-up is needed before it can be properly considered as a composition modelling tool.

Whether or not these beginnings will be refined into a successful model is a difficult question. Usually, successful models are those that suppress as much detail as possible, but this model incorporates the full complexity of Turing machines. As computer scientists are fond of repeating, Turing machines are fully capable computers, and one wonders whether any real simplification is achieved by using them in the model rather than, say, DEC VAX computers. If the Turing machine represents something like the computational limitations of an observer, this objection is clearly invalid, but that does not appear to be the case in this model.

7 NONDETERMINISTIC SECURITY MODELLING

History abhors determinism but cannnot tolerate chance.
Bernard De Voto,
The Course of Empire (preface),
1952.

Science repulses the indefinite.
Claude Bernard,
Introduction à l'Étude de la Médecine Expérimental,
1865.

7.1. INTRODUCTION

David Sutherland's 1986 paper [Sutherland86] introduced the nondeducibility definition of security which, among its other attributes, appears to have been the first such formal definition that could be applied to nondeterministic systems. There are, however, no indications in the paper that the author's efforts had been directed specifically in that direction, or that the achievement was recognized as being of particular importance.

Since that time, nondeterminism has gained an increased following among security theorists. Nondeducibility sparked the development of many new models and modelling methods in which nondeterminism is held to be a fundamental concept, reflecting the growing conviction among some researchers that the ability to represent nondeterministic systems is essential to any modern formal definition of security.

This chapter deals with several questions about nondeterminism in security models. Foremost among these is the validity of this emerging belief in the indispensability of nondeterminism. Our thesis is that, in fact, nondeterminism is neither necessary nor desirable, and that existing nondeterministic models fail to adequately address the security modelling problems cited as evidence supporting the need for nondeterminism.

7.2. NONDETERMINISM VERSUS RANDOMNESS

The original referent of *determinism* is a philosophical doctrine that denies the existence of free will. When the word was adopted for various mathematical uses, it spawned the antonym *nondeterminism* , which acquired an interpretation slightly different from ''free will''.

Formal language theory uses mathematical structures called *nondeterministic automata*[1] . The *nondeterministic Turing machine* encountered in the study of computational complexity is a close relative. They, and their deterministic counterparts, are idealized machines which, when presented with some input, proceed through a sequence of ''moves'' (state transitions) and possibly eventually halt. In the nondeterministic automaton, the state transition function maps the current state and input symbol into a set of possible moves rather than a single move. The automaton, however, does not randomly select one of these possibilities (a machine that does is properly called *probabilistic* rather than nondeterministic). Instead, it effectively enters all of those next states, thereby simultaneously performing all possible computations on its input. This can be viewed in various ways: that the current state is really a set of states, or that the machine duplicates itself and each duplicate follows a different transition, or that it somehow ''knows'' which transition will lead to a successful computation and follows that one.

In contrast, researchers in formal security models have tended to use the term ''nondeterministic'' to refer to machines whose operation contains some elements of randomness. ''Probabilistic'' would be a better name, but for those models that represent the variations without specifying probabilities it might be less misleading to use one of ''random'', ''stochastic'', or (the author's favourite) ''aleatory''.

However, rather than trying to correct an established misuse, the current security modelling practice will be continued here. Readers familiar with the term ''nondeterministic'' as it is used elsewhere are encouraged to read it as ''probabilistic'' wherever it appears in the balance of this document.

7.2.1. The History of Nondeterminism

As mentioned in the introduction, the first formal definition of security to be applicable to nondeterministic systems was nondeducibility [Sutherland86]. Briefly, let Ω represent the set of all possible system executions. These may take the form of event traces. Let $visible_s(\omega)$ represent the information about the execution $\omega \in \Omega$ that observer s is permitted to see. Also, let $secret_s(\omega)$ be other information that s is not permitted to know. The system is nondeducibility-secure (with respect to s) if, for each pair of executions

---

[1] [Aho72], p. 113.

$\omega_1, \omega_2 \in \Omega$, there exists an $\omega_3 \in \Omega$ with the properties

$$visible_s(\omega_3) = visible_s(\omega_1)$$
$$secret_s(\omega_3) = secret_s(\omega_2)$$

One possible instantiation of the model is that the Ω are event traces in which each event is assigned a sensitivity. $visible_s(\omega)$ is the subsequence of events whose sensitivities are dominated by subject *s*'s clearance, and $secret_s(\omega)$ is the subsequence of *input* events of higher or incomparable levels. The model assumes that the only source of high information is high inputs. The definition is basically a requirement that every possible observation made by *s* must be compatible with every conceivable secret. This prevents the observer from deducing the secret based on his observations.

Some time after the introduction of this definition, researchers began looking rather carefully at its handling of nondeterministic systems. That it permits them is clear: there is no requirement that ω_3 be unique, its visible and secret parts may be freely interleaved, and there are no constraints at all on other events (i.e. high outputs). However, it has several problems that relate to the current discussion [2].

First, nondeducibility defines security as a ''liveness'' property. This is a requirement that a system must be capable of behaving in a certain way (ω_3, to be precise), rather than a prohibition against specified behaviour. If the actual system displays only some of these modelled traces, for example by being more deterministic than its model, it may fail to meet the security requirement. This may happen if the model's nondeterminism represents underspecified behaviour that is fixed in the implementation. This difficulty renders nondeducibility unusable for modelling underspecified systems, as one might wish to do early in the design phase.

Secondly, and perhaps more seriously, nondeducibility makes the most optimistic assumptions possible about the statistical properties of the random (or nominally random) behaviour of the system. It calls a system secure as long as there exists at least one execution ω_3 for each pair of ω_1 and ω_2, no matter how unlikely that ω_3 may be. The following example from [Thomson88] is illustrative:

> *Two sites are connected by a communications line that has a uniform bit error rate of 10^{-9}. At one site, highly sensitive information is fed into the channel, while at the other end an uncleared observer (call him the **viewer**) watches what comes out.*
>
> *There is a small probability, but a nonzero one, that an arbitrary input may be corrupted by the channel noise to form any specified*

[2] Other points of contention, such as its incomposability, or its hiding only inputs, are not pertinent here.

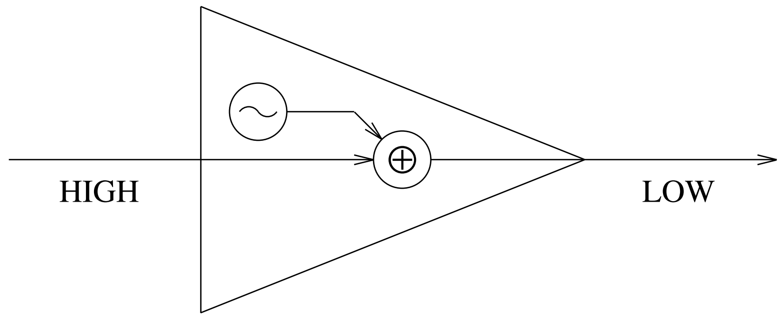


Figure 7.1: Communications Channel with Internal Noise Source.

output, so it is the case that every possible observation is compatible with every input. This system, then, is nondeducibility-secure, and remains so no matter how low we make the bit error probability, as long as it is greater than zero.

This lack of robustness occurs because the model represents only the range of the noise, and because it is concerned only with absolutely certain deductions by the observer. Statistical inferences are not considered.

In the following years, researchers pursued definitions of security that were preserved by interconnection. It was believed that the composition of subsystems entails nondeterminism [McCullough88b], and so McCullough's hook-up security [McCullough87] and related restrictiveness [McCullough88b] properties admit nondeterministic behaviour. These, along with forward correctability [Johnson88] (a variation on the hook-up property), and output security [Guttman88], are composable properties that permit nondeterminism but do not address either of its problems any better than did nondeducibility.

More recently, researchers have attempted to solve the optimism problem by adding some statistical information to the model. [Wittbold90] describes what is in essence a stateless communications channel with noise. The input, output, and noise signals can each assume a range of values. By appeal to information theory, the authors make the point that the capacity of the channel is zero only if the probability distribution of received symbols is independent of the transmitted symbol. Our earlier example of this kind would be found to be insecure by this definition because, if the transmitter was sending a ''0'', the receiver would see a ''1'' with probability 10^{-9} and a ''0'' with probability $1-10^{-9}$. If the transmitter sends a ''1'', the receiver now sees a ‘‘1’’with probability $1-10^{-9}$ and a ‘‘0’’with probability 10^{-9}.

In [Gray90], the author begins with the state machine described in [McCullough88b]. This is an asynchronous machine whose state transitions may occur on inputs, outputs, or internal events. Both outputs and next state are probabilistic; this is captured in the model by defining the relation $T \subseteq S \times E \times S$ where $t = \langle s_1, e, s_2 \rangle \in T$ means that the machine in state s_1 may engage event e and thereby move to state s_2. There may be several output events possible in any given state, and also several possible result states. McCulloughs security definition requires that high input events must not alter the set of possible low-level events that follow. Gray augments this by further requiring that the relative probabilities of the following low level events must not change.

The *flow model* of [McLean90] also seems to be loosely based on this same idea of probability invariance, but the presentation is rather sketchy and not easily compared to other examples [3]. Rather than being concerned with high and low level information, this model deals directly with high and low labelled objects. Instead of events and event sequences, it deals with states (object-value assignments) and so cannot be applied to a stateless system such as a communications channel. It also the probability of these states and of sets of states, which makes little sense unless the inputs are fixed or their statistical properties are known, neither of which appear to be the case here. Still, it does seem to contain the essential idea that the probabilities of different low-level observations must not depend on any high-level information.

It is clear that the inclusion of probability in these models improves their handling of nondeterminism. If we view a system as a communications channel, where a transmitter interferes with a receiver’s observations subject to perturbation by noise, the usability of the channel depends on the construction of the channel (which determines how effectively the noise is able to perturb observations) and on the amplitude and unpredictability of the noise. Nondeducibility models the channel construction and the (peak) noise amplitude, but assumes minimal predictability. Adding explicit probabilities tightens up the definition by at least partially characterising the predictability of the noise.

Unfortunately, these models are all still optimistic. They ignore the fact that there is more to a random process than a probability distribution. The only full characterisation of a signal is the joint probability distribution of its value, all previous values, and the current and previous values of all other signals in the system. Wittbold recognizes this danger and avoids it by

---

[3] McLean compares it to nondeducibility, but the comparison is based on an invalid interpretation of Sutherland’s high-level information functions as the contents of high-level files. Nondeducibility depends on the actual sensitivity of information, not on the (possibly higher) label of its container.

assuming that ''... all symbol transmissions experience independent corruptions'', but it is by no means clear that this assumption is valid in the networked and distributed systems that their model is intended to serve.

This point will be discussed further, but first it will be useful to formally define nondeterminism and to consider what exactly are the sources of this ''random'' behaviour that seems so badly in need of modelling and so difficult to handle adequately.

7.2.2. Definitions of Nondeterminism

The simplest definition of nondeterminism useful in security modelling is one that applies to state machines:

> *A state machine is* ***deterministic*** *if its next-state and output mappings are ordinary single-valued functions. It is* ***nondeterministic*** *if either mapping is probabilistic.*

The heart of the Bell and LaPadula access control model [Bell74] is a deterministic synchronous state machine. Goguen and Meseguer's non-interference model [Goguen82], though it defines security rather differently, is also constructed around such a deterministic machine. These are both synchronous machines because they operate in well-defined cycles where they accept an input, produce an output, and change state.

Synchrony is adequate for many simple systems, but more complicated machines often undergo complex internal state changes between an input and any corresponding output, and often are able to generate outputs independently of inputs. A more faithful representation of the sequencing of operations is available in the form of event traces. Event traces are sequences of events representing system executions. The events may include input events, output events, internal events, and possibly composite events. The passage of time may even be crudely modelled by the inclusion of ''clock tick'' events. With this formalism, a system is characterised by the (usually infinite) set of event traces that represent valid executions of the system.

It is even possible to combine the state machine and event trace formalisms, as McCullough does [McCullough88b] to create a state machine that is excited by generalised events rather than only by inputs - i.e., its state may also change as the result of producing an output or undergoing an internal event. His model is nondeterministic both because, in general, more than one kind of event can occur in a given state, and because the state entered after the event is not uniquely defined.

To make some sense out of all these competing definitions, it is desirable to settle on a definition of nondeterminism that applies to a wide variety of system representations. We choose the event trace as the lowest common denominator of these representations. For synchronous state machines, the traces considered are the sequences of input-output pair events that can be generated by the machine. For other machines, the events of the trace may be more general than this.

The essence of determinism, and the key to defining its opposite, is that a system's response to any given set of inputs is uniquely defined. We can distinguish three candidate definitions of determinism for event traces. Let T be the set of traces that characterise some system, and let $I(t)$ and $O(t)$ be the subsequences of input and output events in the trace $t \in T$, respectively.

Definition 1:

A set of traces T is ***strongly deterministic*** *iff, for any two traces t_1 and t_2, $I(t_1) = I(t_2)$ implies $t_1 = t_2$*[4].

This very strong statement is what results if ''response'' is interpreted to mean the entire execution trace, including not just all the events but also their relative ordering and any timing information that may be contained in the trace. The previously encountered synchronous deterministic state machines generate strongly deterministic traces.

If, instead, ''response'' is interpreted to mean the sequence of outputs, the following looser definition results:

Definition 2:

A set of traces T is ***input-output deterministic*** *iff $I(t_1) = I(t_2)$ implies $O(t_1) = O(t_2)$.*

Here a particular input sequence may correspond to several executions that differ in their unobservable internal events or in the precise interleaving of inputs and outputs. This would be useful to describe pipelined systems, or asynchronous systems with unbounded buffering. A deterministic state machine followed by a stage that delays the outputs generates traces of this type.

Definition 3:

A prefix-closed set of traces T is ***eventually deterministic*** *iff for all pairs of traces $t_1 = \alpha\varepsilon$ and $t_2 = \alpha\gamma$, where α is some event sequence, and ε and γ are non-input events, implies $\gamma = \varepsilon$.*

This last definition is used in the discussion of rated event systems in [McCullough88a]. If an output or internal event occurs at some point in a trace, it must be unique, but if a new input occurs first then an entirely different trace may result. This captures the notion of machines that will act in a predictable way if given enough time to produce their outputs (hence, ''eventually''), but may act differently if a new input happens (eg. someone

---

[4] This definition and the next are really too simple to be useful, because they do not permit T to be prefix-closed. Example: let t_1 be $\langle i_1, o_1 \rangle$ and t_2 be its prefix $\langle i_1 \rangle$. But, if t_2 is a trace, then T cannot be either strongly deterministic nor input-output deterministic. This may be corrected, but the definitions become rather more complicated. See p. 129 of [Thomson88] for an instance of this.

pressing the "Interrupt" button) before the output does.

The first definition is stronger than the second, which in turn is stronger than the third. Practical asynchronous systems we might wish to characterise as deterministic are probably closer to satisfying one of the latter two definitions than the first, since they do not have rigid schedules of input deliveries. For each of these types of determinism there corresponds a definition of nondeterminism that is satisfied when the implication fails.

7.2.3. Sources of Modelled Nondeterminism

Many arguments are offered for the desirability and even necessity of nondeterminism. These may be grouped according to the nature of the effect that the nondeterminism in the model is intended to represent.

Type 1 effects are those that are the manifestations of true stochastic processes, as addressed in [Gray90]. These exist in everything we build, but in most applications a substantial part of the effort of engineering design is directed toward minimizing their effects. Rarely, devices are designed so that their operation intentionally contains a significant random component. A random number generator built around a noise diode would be an example. Such devices are the exception rather than the rule.

Processors, storage devices, and communications systems are all designed to closely approximate deterministic operation in spite of the ubiquitous thermal and other natural noise sources. As a result, although the effects of this noise cannot be entirely eliminated [5], they may be rendered so rare (eg. a disk with error correction may exhibit an uncorrected error rate of only 1 in 10^{12} bits) that it may be more appropriate to threat these residual noise effects as component failures rather than as characteristics of normal system operation.

Related to true randomness (we might call these Type 1.1) are the pseudorandom systems which, though deterministic, mimic by design the statistical properties of true random processes. Some are better at this than others, but they are all subjected to some amount of statistical analysis to see just how good their approximation is. Since they are much more convenient to use than true random sources, they find application in everything from simulators to Monte Carlo solvers, to packet retransmission delay algorithms, password generators, encryption devices, and as noise sources to limit the bandwidth of covert channels.

Type 2 nondeterminism is the kind resulting from underspecification. It is reasonable that designers would wish to construct a security model of a

---

[5] At least, not in finite time. C.E. Shannon's celebrated coding theorem requires infinite signal durations to achieve totally error-free communication in the presence of noise.

system early in its design. At that stage, there may be many details that are not yet determined, but which may affect operation in some detectable way. Still other operational characteristics may not actually be known until a prototype device is built and can be instrumented during operation. In both cases, a modeller may resort to nondeterminism to account for the variability attributable to these unknown or unresolved factors.

Type 3 is nondeterminism resulting from modelling abstraction. A security modeller may choose to omit the particulars of detailed or complex deterministic processes in the system, or may employ a modelling scheme that is incapable of representing these processes in sufficient detail to expose their deterministic nature. In this way, behaviour that is actually deterministic may appear as nondeterministic in the model.

One example of this is the CPU scheduling in a timesharing computer. Typically, the algorithms used are not terribly complex, but the inputs they require may be next to impossible to provide in the model. At each rescheduling point one must know (at least) which processes are ready to run, and, having selected one, whether it will first exhaust its quantum, suspend itself, or be pre-empted by another higher priority task. This much information can only be acquired by full simulation of the system in question, which is infeasible. A practical alternative is to model only the gross characteristics of the scheduler (i.e. non-starvation) and to use nondeterminism in the model to subsume the observable variability in execution speeds.

Another example is a system with finite input buffers that may be overrun, losing or rejecting inputs if too many arrive in a burst. The system may then exhibit different behaviour in response to a given sequence of inputs depending on how rapidly they arrive compared with how quickly the system deals with them. This could be modelled deterministically if we include timing in the model and explicitly model the input buffer and the internal events that drain it. However, it is also tempting to suppress this detail and achieve a simpler model with no timing, no input buffer, and a dose of nondeterminism to account for the otherwise unexplainable loss of inputs. Here, the more precise model may be possible but awkward.

Various scenarios presented in the literature can be seen to be instances of this model-inspired nondeterminism. [McCullough88b] contains some of these, including a version of the finite buffer overflow example. It also contains a second example of two components connected so that their outputs merge [6]. Even if the components are individually deterministic, unless their models include timing information it will not be possible to predict how their outputs interleave.

---

[6] Actually, it is sufficient that their outputs be connected to the same third component, and do not have to be merged into a single stream, for the demonstrated situation to occur.

McCullough presents these examples as evidence that the composition of deterministic systems entails nondeterministic effects, and concludes that a deterministic model cannot serve as the basis of a composable security definition. However, it is the case that the actual system is deterministic, and the apparent nondeterminism is an artifact of the model and of its limitations. The modeller trades off precision for simplicity, and, in so doing, creates the impression of randomness.

7.3. PROBABILISTIC MODELS REVISITED

The practice of modelling deterministic detail as randomness is widespread in the field of computer and communications engineering. Queueing theorists commonly model processors, disks, and even users as random processes, and often as Poisson processes (even if the approximation is known to be poor) because doing so may greatly enhance the tractability of the problem.

This is an accepted approach because approximations to average-case performance, even order-of-magnitude approximations, are appropriate where these models are used. Designers of real-time systems do not use these approximations, because they require bounds for worst-case performance. The performance estimates they produce, when they err, must do so on the side of pessimism. We believe that security modelling more closely resembles design of real-time systems, in that if the models do not accurately reflect the security of the modelled system they must err on the side of pessimism. Moreover, security models must bound the worst-case operation of a system assuming intelligent and malicious attacks, which is an even stricter requirement than that faced in real-time system design.

Nondeterministic models proposed in the literature do not have this property. Even models that include probabilities may be optimistic, because they assume that the ‘‘random’’ variations in the machine’s operation have the most favourable other statistical properties, namely that every variation is statistically independent of all others.

On those rare occasions where real randomness is being modelled, it may be possible to determine accurate probabilities for a model and to justify the assumption of statistical independence. This can also be done for the pseudo-random case, if the properties of the generator can be assured.

Extending this approach to the cases where nondeterminism is only an approximation may lead to significant errors. The ‘‘noise’’ may correlate with its own previous values, as in a low-quality pseudorandom number generator. Also, noise sources modelled as independent may correlate with one another, particularly if they are manifestations of the same unmodelled mechanism. Finally, if the modelling process sweeps mechanisms and interactions under the rug of nondeterminism we cannot develop any assurance that those effects are immune to manipulation by a malicious user. In each of these cases, a probabilistic model will give an optimistic assessment of the system's security.

7.3.1. Quasi-Determinism

In [Thomson88] and [Thomson89] it was proposed that this problem be dealt with in a very different way. Rather than adding more and more statistical detail to nondeterministic models in an attempt to reduce their optimism, Sutherland's nondeducibility model was modified to represent the effects of nondeterminism in a pessimistic way, and to do so without all of the complexity inherent in the probabilistic approach. This approach, which we will call *quasi-deterministic modelling*, consists simply of building a fully deterministic model with extra inputs, inputs that must be capable of perturbing the operation of the system to the same degree that the so-called nondeterminism would.

Extraction of randomness to form an extra input is an established modelling technique for control systems [Kumar86], where it permits the use of conventional techniques to analyse the system. In security modelling, it allows us to model the effects of nondeterminism no matter what its origin, while retaining the simplicity of a deterministic model.

If we do this to our communications channel example, we will model the system as shown in figure 7.2. Since the system is now deterministic, it may be modelled using nondeducibility without incurring either of the drawbacks described earlier (of course, other modelling methods could also be used).

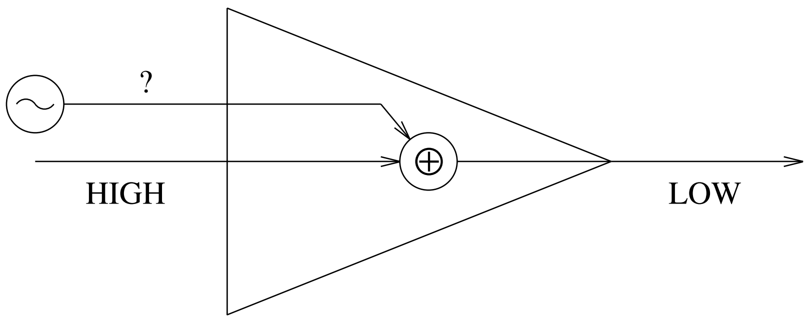


Figure 7.2: Communications Channel with Noise Source Externalized.

The precise effect of formally creating the new input depends on how it is treated during the modelling.

If we consider the noise input to be a high-level signal, it will be included in the $secret_s(\omega)$ function for the low-level observer s. Since $secret_s$ now determines both inputs to the deterministic exclusive-OR function, it is no longer independent of $visible_s$, and so the system is found to be nondeducibility insecure, which agrees with our intuition.

The reader may object in the case of the present example, because there is no compelling reason to believe that the noise signal should be considered high. If, instead, the modeller considers it a low-level input then, as there is no formal requirement that it be hidden from the observer, it is reasonable to take the pessimistic view that it is part of $visible_s(\omega)$. With this do the system will again be found insecure.

7.3.2. Controlling Pessimism

Note that no reference has been made to the statistical properties of the signal in either of the previous cases. Both analyses are therefore *pessimistic* — they find the system to be insecure no matter what the characteristics of the noise signal.

There is a third option for analysis in cases where the pessimism of the first two is known to be excessive, or where a more detailed analysis of the system is desired. This might be appropriate in situations where the statistical nature of the interfering noise is well known. It involves characterising the system's operation as a function of the extra signal, regarding it as a parameter rather than as an ordinary input with an assigned level. Statistical knowledge concerning the noise input may then be used with the parametrized model to derive statistics relating to the system's execution.

Since the model will now be dealing with probabilistic issues (though the system model is deterministic, the noise inputs are not) a security definition is required that prohibits probabilistic inference. Non-deducibility is an attractive starting point for formulating such a definition.

Let Ω be a deterministic set of worlds, and $visible_s(\omega)$, $secret_s(\omega)$, and $noise(\omega)$ be information functions such that $noise(\omega)$ is a value of a random variable N. Then, the system is ***quasi-deterministically non-deducibility secure*** with respect to $visible_s$ and $secret_s$ if the following holds:

For each observation $v \in \{visible_s(\omega),\ \omega \in \Omega\}$ there exists a unique probability p such that for all secrets $k \in \{secret_s(\omega),\ \omega \in \Omega\}$

$$p = \sum_{\substack{\omega \in \Omega, \\ visible_s(\omega) = v, \\ secret_s(\omega) = k}} \mathrm{Prob}(N = noise(\omega))$$

The sense of this definition is that all secrets must be equally likely given any observation. In other words, the probability of a noise sequence that will allow a secret to coexist with some observation must be independent of the secret.

In our communications channel example, the memoryless nature of the channel means that the functional dependency can be expressed as

$$out_i = in_i \oplus noise_i$$

where x_i is the ith element of the n-long bit sequence x^n. We identify the n-bit sequences in^n as the secret, out_n as the observation, and $noise^n$ as the random noise. If the individual noise values $noise_i$ are independent and equiprobably 0 or 1, then given any n-bit observation and secret, the probability of a noise sequence necessary for their coexistence is $1/2^n$, which is independent of the secret sequence. The system is secure.

If the noise signal samples are again independent, but take the value 1 with probability only 10^{-9}, the system is insecure. A simple counterexample with $n = 1$ has $out = \langle 0 \rangle$. Coexistence with $in = \langle 0 \rangle$ requires $noise = \langle 0 \rangle$, which occurs with probability $1-10^{-9}$, while coexistence with $in = \langle 1 \rangle$ requires $noise = \langle 1 \rangle$, which occurs with probability 10^{-9}. Since the probabilities vary for different secrets, the system is insecure.

Consider finally the case where the noise source has the behaviour

$$\begin{aligned} noise^i &= noise^{i-1} \text{ with probability } 0.9 \\ &= 1-noise^{i-1} \text{ with probability } 0.1 \ . \end{aligned}$$

The probability of a single noise sample being a 1 is $1/2$, as in the first case, but the samples are no longer independent. This noise process generates alternating strings of zeroes and ones, with a mean burst length of 10 samples. This is like having low-frequency bandlimited noise in a communications channel. This system is also insecure, as can be seen from an example using 4-bit sequences. Let the observation be the string $\langle 0,0,0,0 \rangle$, then a secret input of $\langle 0,0,0,0 \rangle$ requires a noise sequence of $\langle 0,0,0,0 \rangle$ which occurs with probability $1/2 \times 9/10 \times 9/10 \times 9/10 = .36$. The secret input $\langle 0,1,0,1 \rangle$, however, requires a noise sequence of $\langle 0,1,0,1 \rangle$ to coexist, and the probability of this is only $1/2 \times 1/10 \times 1/10 \times 1/10 = 0.0005$. Note that the probabilistic security models, which assume independent perturbations, would not be able to distinguish this case from the first one and so would erroneously consider this system secure.

7.4. EVALUATION OF NONDEDUCIBILITY

There are various sources of variability in the execution of realistic systems. Nondeducibility and its immediate successors represented this variability, but did so in a way that could give very optimistic assessments of security. Probabilistic models, where every transition is assigned a probability, are appropriate only where the variability is (or very closely resembles) true randomness and where meaningful probabilities may be assigned to the transitions. Where this is not the case, more detailed modelling may uncover the deterministic mechanism behind some variation but the increase in model complexity can quickly become overwhelming.

Quasi-deterministic modelling appears to be applicable in all these cases. Where variation exists but cannot be statistically characterised, the model used is naturally pessimistic. On the other hand, if the statistical nature of the random effects can be realistically represented then the opportunities for probabilistic inference can be accurately detected.

7.5. WHAT IS SECRET?

Early information flow models including noninterference [Goguen82], nondeducibility [Sutherland86], and restrictiveness [McCullough87], identify the sensitive information in a system as the sequence of high-level inputs it receives, and define confidentiality as the prevention of disclosure to unauthorized observers of those inputs. This definition subsumes detail such as the existence of labelled objects — if an object's contents are derived from past high-level inputs then they must not be divulged, and as far as the model is concerned the object's label is simply a bookkeeping device used to achieve this protection by an implementation [7]. Similarly, low observers cannot have access to outputs that are derived from high inputs. Thus, high inputs are explicitly protected, while files and outputs may be consequentially protected to the extent that they do or may inherit sensitivity from high inputs used to create them.

Joshua Guttman and Mark Nadel challenged this practice in [Guttman88], arguing that in many systems high inputs are not the only sources of sensitive information. They suggest that a valid definition of security should also explicitly protect high-level outputs, to ensure that any high-level contents that are not derived from high inputs will also not be divulged. They also contend that the interleaving between high-level inputs and low system activity may be sensitive and should also be explicitly protected.

Since that paper was published, there has been surprisingly little discussion of the topics it raised. Some published material references it and even accepts its assertions, but there is little or no work to be found that actively either supports or refutes them. In this chapter, however, it will be demonstrated that the issues raised by Guttman and Nadel have not yet been

---

[7] Implementations usually protect files and outputs which are labelled high, whether or not the information in them is actually derived from high inputs. The models permit this extra protection, but do not require it.

appropriately resolved, and that it would be premature to attempt to ‘‘close the book’’ on them.

7.5.1. Output Security

In support of their claim of innate output sensitivity, Guttman and Nadel present several examples that are perhaps not as convincing as they might be. Their third example contains a decryptor, and it is argued that the output is sensitive although the inputs are not. This is clearly not the case, since one very important input is the decryption key, which is very sensitive indeed. In this case, at least, the conventional approach will suffice.

However, this does not deny the validity of their claims as much as it illustrates the difficulty of generating good, non-toy examples. It is certainly true that the outputs of real systems may be sensitive despite their independence from high-level inputs. There are several ways that this can happen:

1) Nondeterminstic systems may spontaneously generate or modify high-level outputs. The outputs are then not fully determined by inputs, and at least in the information-theoretic sense contain more information than the inputs. It is conceivable that this information might be sensitive, perhaps not intrinsically but because of the way it will be used. For example, the government’s revenue department might use a random-number generator to select taxpayers for income tax audits, and may validly desire to prevent leaking the selections and tipping off the lucky winners.

2) Time and computational resources used in producing a result, even if it is ultimately derived from non-sensitive data, can constitute added value that is worthy of protection. This might occur in the national security setting, as in the results of a sophisticated analysis or simulation based strategic projection, but may be more familiar in the commercial sector where the sensitivity of information is directly related to the commercial advantage enjoyed by its holder. But perhaps the classic example is a code breaking machine that generates two very sensitive 100-digit prime factors from its input, a rather mundane number 200 digits long.

3) Sensitive information built into the structure of the system, such as a pre-loaded database or hardwired key, may also contribute information to outputs. Sutherland explicitly disallows this in his non-deducibility definition, since he assumes that the construction details of the system are public knowledge. Other trace-based definitions make the same assumption, although they may fail to recognize it. Systems of this type may be modelled by assuming the presence of phantom past inputs responsible for creating the database or installing the key, but this practice is somewhat unnatural and nonintuitive and therefore may contribute to modelling errors.

4) As a distinguishable variation of previous cases, if secret algorithms are employed to process data then the sensitivity of the results may be affected. This is similar to the case 2, except that here the factor contributing to the value of the outputs is embedded high data or development effort rather than computational effort. It is also similar to case 3, except that the sensitive technology is embedded in a program rather than in preloaded or preconfigured data, and also that the program need not be initially present in the system. Actually, a consistent interpretation of most models would regard a computer system as a machine-instruction interpreter, and a program as part of the data input to that system. Then, just as in the general case of a computation driven by high-level data, the outputs would naturally inherit the sensitivity of that data. However, there is evidence that this interpretation is not sufficiently obvious (eg. [McLean90] appears to consider it novel) to be widely practised.

Guttman and Nadel propose a security definition that protects these other sources by explicitly protecting all aspects of high-level outputs except those that can be traced to low inputs. The exception is made to permit write-up. McLean recommends a similar style of output protection in his FM model [McLean90].

It is the reputed advantage of these definitions that they protect all the information in high outputs that comes from the ‘‘other sources’’. Unfortunately, this characteristic is also their greatest fault: any information generated within the system, if it affects high-level outputs, will be considered high-level. This happens automatically and ungovernably, despite any reasonable wishes to the contrary on the part of the system designers or users, because it is inherent in the notion of output security.

As a result, no system that is to be judged secure by these models can generate unclassified random numbers, or process transactions against an unclassified database, unless its security policy prevents those activities from affecting high-level outputs. This amounts to the elimination of write-up occurring within the system, and Guttman and Nadel’s model does not, in fact, permit write-up to occur unless the low input is received directly from a user. The effect of this rule depends on the granularity of decomposition during modelling, but presumably would prevent high-level users from reading low files fetched over a network or from magnetic tape.

It is difficult to see how restrictions like these contribute to the security of a system. It seems more likely that the formal security definition that inspires them is too aggressive, and attempts to protect more than really deserves protecting. An improved definition should allow finer discrimination of information and more accurate description of sensitivity than this.

7.6. UPGRADING AND DOWNGRADING

Popular security models also have difficulty encompassing systems where the sensitivity of information changes, or where special circumstances render a configuration secure that would ordinarily be considered insecure. Often, any processes by which these changes occur is required to be analyzed outside the model by some alternate, special-purpose technique. Information-flow-based security definitions may not permit representing these flows even if they have been separately approved. Although it may not be reasonable to expect a general security model to handle verifying the security of these functions, it is still desirable that the result of verification be representable in the model.

Upgrading is not particularly difficult to handle, since it may usually be modelled either as simple write-up, or as computational added value according to case 2 of the previous list. A third possibility is when two or more non-sensitive data items are selectively bound together (eg. a picture of the Statue of Liberty and the phrase ''Missile Site'') forming a high-level compound, but since this binding is not spontaneous there must be other inputs to the process that can be identified as supplying the sensitive information.

Downgrading is a much more difficult problem. Information-flow models do not admit that a low-level output can securely depend on high-level inputs. In practical systems, there are several ways in which this can happen.

Downgrading can happen as a filtering operation, in which formally high-level data is sanitized, often manually, extracting low-level information from it. A reasonable view to take in this situation is that the input actually consists of intermixed high and low-level information, and that rather than saying that the output information has changed level it would be more correct to say that its overclassification has been corrected. Since this process ultimately depends on the correct functioning of a human, full formal verification is probably too much to expect. Still, a model would be more complete if it could at least represent the (assumed) secure functioning of such a downgrader. Perhaps a recognition in the model of the difference between sensitivity and label might permit this representation.

A rather more difficult downgrading event is one in which the actual sensitivity of information does change. This is fairly common with classified material that becomes less sensitive with the passage of time [Williams88], and may even be mandated for data subject to freedom of information laws. It is different from the sanitizing downgrader both because the information really does change level, and because the event that triggers the downgrade may be as mundane as the changing of the calendar, where previously it was a carefully considered decision of a security officer. [Sutherland89] describes a notation that can express changing levels of this type and sketches a security definition that employs it.

Encryption is a process in which high-level inputs (data and key seed) produce a low-level output. If it is a public-key system, then both the output and the input key are low. Information flow considers these insecure, because the output is affected by the inputs (noninterference) or is correlated with particular input values (nondeducibility). These are prohibited because, in principle, an observer could invert the function and deduce something about the sensitive input, but in practice the inversion is sufficiently difficult that the deduction is impractical. Cryptologists can distinguish between secure and insecure encryption systems because they explicitly model the computational capabilities of the observer. Adding that full capability to a general model would probably be excessive, since we wish only to describe systems containing encryption rather than prove the encryption secure.

Looking inside the encryptor, we may find one unit that expands the supplied key into a semi-infinite string of symbols and a second unit that combines this key stream with the data to produce the encrypted output. The expanded key is highly sensitive, since its possession would enable the simple decryption of the ciphertext. The unit that combines the two high streams is yet another difficult case of downgrading, because it combines two high-level signals using a simple and easily invertible function to form its low-level output. Information flow calls this insecure as well, and appeals to computational complexity are not available.

Intuitively, the key stream is not innately sensitive, and we permit an observer to deduce possible pairs of data and key stream symbols because its randomness prevents deduction about the data alone. However, conventional information flow will accept this only if we treat the key stream as a randomizing influence, and not as a signal like other signals. But, if we use this approach, we lose the ability to distinguish between a secure system where the key stream is protected as high-level data and an insecure one in which it is not.

7.7. WHAT TO DO?

So far we have an extensive wishlist but few leads on how to accommodate it all. It is not even clear at this point whether a sufficiently comprehensive security definition is feasible. We wish to handle as many of these difficult cases as possible, but without significantly increasing the complexity of modelling simpler systems. That is, any complexity added to deal with these difficult cases should, as far as possible, affect only models of systems that display the troublesome characteristics.

Some of the discussion in the previous sections suggested the ideas of the transmission of sensitivity and of clarifying the distinction between sensitivity and label. We begin by looking a little more closely at these issues.

7.7.1. Sensitivity Flow

Earlier it was remarked that many early security models assumed that the sensitivity of high inputs is inherited by the high outputs they are used to produce. As information flows from the inputs to the derived outputs, it carries its sensitivity along. The explanation for this is that disclosure of the outputs would provide some information about the inputs, and therefore must be avoided.

Often, though, the ‘‘flow’’ of sensitivity is really in the other direction, originating in the use of some signal and propagating back to its source. It may be fair to say that information comes from sources, but sinks are the origins of sensitivity.

In some sense, of course, all classified information derives its sensitivity from its use. More precisely, the sensitivity is a consequence of the potential for abuse if the information should be improperly distributed. However, sensitivity can also be traceable to a permitted use. This is particularly obvious in cases where the sensitivity of an output depends on how it is used rather than any meaning it might possess.

In the dissected encryptor example, the pseuorandom generator output has no inherent meaning, and so cannot be said to be sensitive for that reason, but is very sensitive if used to encrypt other high-level data. And, therefore, the key seed input to the generator must also be protected, because someone who knew it could duplicate the key stream output. This is an example of sensitivity propagating from an output back to an input.

Occasionally, sensitivity may also spread from one input to another. The key stream input had to be protected as high-level in order to protect the high-level plaintext data — therefore it inherits its sensitivity from another input. It seems that sensitivity can originate in a small number of meaningful signals, and propagate backward, forward, and sideways through system components to other signals.

Informally, at least, we can recognize sensitivity propagation by distinguishing between signals that are *inherently* sensitive and those that are *consequentially* sensitive. The sensitivity of inherently sensitive signals is imposed from outside the system. In order to protect those signal from disclosure, some collection of other signals must also be assigned a high sensitivity, and these are the consequentially sensitive ones.

Sometimes, a system may have more than one satisfactory assignment of sensitivity levels to signals. The encryptor submodule that combined the key stream with high-level data had to have its second input be high level because its output was low. Note, however, that if a functionally identical component were used such that its output were high, the second input could be left low without incurring any compromise. Knowing the functional specification of the component plus the fact that one of its inputs receives high-level data is not sufficient to uniquely determine the correct labels to apply to its other ports. The choice of the right assignment here depends on the externally imposed constraint that the output must be low. Our interpretation of this is that a security model should not dictate the sensitivity of information, but should allow the modeller to represent some proposed assignment together with a description of the system functionality, and from this the model should indicate whether or not the assignment and function together are consistent with the protection of the inherently secure information.

7.7.2. Labels vs. Sensitivity

Systems that process multilevel information must have some way to distinguish one level from another, whether the separation is maintained through the static configuration of the system or by dynamically tracking sensitivity as data progresses through the system. This is typically accomplished by applying labels to the system's state components (files, variables, etc.) and to its external and internal interconnections. This is convenient when an access control-based definition of security is used to model the system, since the notion of access to labelled objects corresponds very naturally to labelled data containers.

On the other hand, if the actual sensitivity of information is of interest, it is not necessarily accurately indicated by the label affixed to the containing file or port. The contents of a file labelled ''Secret'' are not necessarily Secret, since they may have been written up from some lower-level source. What the label does provide is an upper bound to the sensitivity, a guarantee that the state of the file depends only on information that is ''Secret'' or lower (see [DiVito88]).

An important point to note is that the state of a file includes more than its contents, and sometimes different aspects of its state may have different sensitivies. If a process with access to Secret data writes Unclassified information into a Secret file, we still can not permit an Unclassified user to read it. Not because the contents are too sensitive, but because their existence in that file is the result of Secret-level activity and therefore may encode some information that is Secret.

Even if an Unclassified process writes Unclassified information up into the Secret file, there is still good reason to prohibit subsequent reads by low users. Not because the data is sensitive, nor because it resulted from high-level activity, but because its continued presence in the file is evidence of high-level *in* activity: It indicates that no Secret process has yet overwritten the file. Because its Secret label permits the file to be altered by Secret users, its state will always reflect some Secret information, even if only by its unrealized potential for modification.

What is really ''Secret'' in the last case is not the contents of the file, but the fact that that data is in that Secret file. That is, the data itself is Unclassified but the binding between data and container is Secret since it is subject to Secret-level alteration. To see this, consider that if the Unclassified data were leaked in such a way that its presence in the Secret file were not revealed, there would be no compromise.

What is true for files holds as well for other data conduits. A system output labelled high may carry data that is itself sensitive, or it may carry information that is low but whose presence at that port is sensitive. When components are connected to form composite systems these outputs become inputs to other components, and so inputs that are nominally high-level may also convey information whose actual sensitivity is not high.

Information flow models are concerned with the actual sensitivity of information, not with the labels on its containers. In the absence of better indicators, however, the container label is often assumed to reflect the actual sensitivity of its contents. Thus, non-interference defines the intrinsically sensitive information in the system to be events on high-level input ports. Nondeducibility gives some leeway in its definition, but in practice high-level inputs are a common choice here as well.

Interestingly, the possible discrepance between label and sensitivity does not create problems for these models as they are usually applied. One reason is that the equivalence of label and sensitivity is assumed only for external high inputs, not for high outputs or high internal connections, and so any overclassification that occurs is not ‘‘visible’’ to the model. Compare this with the unusual application of nondeducibility described in [McLean90], where it is interpreted as applying directly to high-labelled objects and, in that incarnation, is found to be flawed. A second reason is that existing information flow security definitions do not depend on ‘‘high’’ information actually being unknown to low observers.

These saving graces may disappear with the emergence of a more comprehensive security definition. Bringing downgrading within the model requires recognizing overclassified data, whether it was overclassified at the time of its introduction to the system or whether its actual sensitivity has changed over time. If a model is to be able to include systems like the dissected encryptor, it must be able to represent the fact that the high-level key stream input really is high-level, and unknown to low observers, and that an upgraded low-level signal is not appropriate here even if it has all the right statistical properties. And, in order to protect high-level information of internal system origin, we need a way to specify sensitivity that does not depend on input port labels but that is more flexible that Guttman and Nadel’s reliance on the output port label.

7.8. PRESCRIPTIONS

The techniques that will be described here allow a security model to deal in some manner with most of the problematic situations listed earlier. Some allow the troublesome mechanisms to be analyzed, while others simply allow them to be represented in a system model after their security properties have been verified by other means.

Unfortunately, these techniques do not afford complete coverage of the problem areas, but for the scenarios they do address they provide flexible and relatively uncomplicated means of handling them in a high-level system-wide security model.

7.8.1. Non-deducibility

We use non-deducibility as the foundation for these improvements because it has the following properties that we feel are desirable:

1) It is based on information flow rather than access control, which we believe to be more appropriate for a fundamental definition of security.
2) It can be instantiated to view components as black boxes, so that their security properties are defined entirely in terms of their external behaviour. Functionally equivalent systems are then credited with identical security guarantees. Although the internal structure of a box may affect our willingness to trust it, this is really a question of assurance rather than of guarantee.
3) Nondeducibility gives some flexibility in the designation of different classes of information that are to be separated. While the definition of noninterference specifically separates the sequence of input events on high-labelled ports from the events on low-level ports, nondeducibility relies on ''information functions ''to isolate the various components of the system execution that should be separated.

Nondeducibility requires the definition of two information functions for each distinct security classification. The first isolates all of the information in a given system execution that is intrinsically sensitive, and the second function represents all information that is directly visible at the classification in question. Nondeducibility security requires that these two functions not exhibit any correlation over the universe of possible system executions.

An event-trace representation of system operation can be employed to allow different aspects of events to be assigned different sensitivities. If ''process p writes data x to file y and receives return code r''is an event, it is clear that the value x conveys information, as does the type of event (i.e. ''write''), and also that the names p and y and the value of r also have some information content. All may have (possibly different) associated sensitivities. In addition, even if all the data in the event is ignored the existence of the event can also contain information, and so also may be sensitive.

In [Thomson88] the visible information function was defined as the subsequence of system events that involve any low-level port. The intrinsically sensitive information was defined as the high-level event sequence filtered by a function called simply the **h** function. Its purpose was to delete all parts of the high-level event stream that were not intrinsically sensitive high-level information. In the example model described in the report, the **h** function left only those parts of an event that were inputs to the system. This permitted what we normally think of as input events to include some inseparable outputs, such as return codes. The event $\langle p, write, x, y, r \rangle$ would be massaged by the **h** function into $\langle p, write, x, y, \varnothing \rangle$, as the non-input part of the event (the return code) is replaced by the placeholder $\varnothing$.

This use of the **h** function is very conventional, since it is only the principle of high-input protection applied to the situation where events may contain both inputs and outputs. However, if we knew more about the actual sensitivities of the components of these high inputs, a more complex **h** function could be more selective in its identification of sensitive information.

7.8.2. Quasi-Determinism

It was argued earlier that an appropriate way to represent nondeterminism in a security model is to define one or more extra system inputs which are assumed to be driven by external random sources. This technique corrected a situation where nondeducibility could be overly optimistic about the noisiness of the nondeterminism.

Another beneficial side-effect of this technique is that it exposes a source of internally-generated information, allowing it to be classified and kept secret (or not) just as any other information source. This method is more flexible than that the Guttman-Nadel approach, because it allows these internal information sources to be treated as having whatever sensitivity the modeller feels is appropriate.

Modelling spontaneous generation or modification of data — the first non-input information source listed in section 7.1 — is an obvious application for this method. Less obviously, it can also be used to model other sources, such as built-in secret information like a database or key (this was the third source in the list).

Non-deducibility assumes that the observer knows which system executions are possible, together with the values that would be taken by all information functions in each execution. In other words, the observer is assumed to know how the system is built, in at least enough detail to be able to enumerate all possible traces. This assumption is incompatible with any secrets in the system construction, and makes it difficult to faithfully model built-in sensitive information. An extra input that conceptually removes the sensitive information from the system allows modelling such a system without rebuilding the basic definition of nondeducibility.

Straightforward use of this technique does not appear to be appropriate for handling the other listed problems. It is tempting to try to use it to model the sensitivity resulting from computational value added, for example by formally adding an extra external high-level input that does not actually affect the output. This can be viewed as injecting high sensitivity into the system and, hopefully, into the output so that it will become consequentially sensitive. However, since the output will not be correlated to the extra high input, nondeducibility will not be fooled by this type of ploy and will not complain if the output is disclosed to a low observer.

This difficulty may be overcome if the security modeller is permitted to be a little less than honest about the functioning of the modelled subsystem. An extra high-level input is introduced, as before, but now it is used as the source of the data contained in any output events. It is probably appropriate to still trigger the outputs by the occurrence of low-level inputs, but the data in the outputs (i.e. the results of the value-added computation that are presumably difficult to duplicate) is modelled as originating in the extra external high-level input. This will result in the output being protected as desired.

7.8.3. Limited Observer Knowledge

Nondeducibility assumes that the observer knows all valid traces and uses this information when making deductions. The extra inputs of quasi-determinism can get around this limitation in some cases, but not in others. Another possible approach is to face the problem directly, and to change the model definition to place limits on the observer's knowledge of the system.

[Marcus88] describes a security model which, although it differs from nondeducibilty in structure, does explicitly represent limits to the observer's knowledge of system operation. Their model is described in terms of variables and their values, and a generalized definition of ''reading'' of variables, defined in [Marcus86], that includes not only direct observation of variables but also any deductions about their contents. In [Marcus88] this definition is summarized as

> There is an *adversary* observing how a given ''open'' variable *v* ''behaves'' during the computation of some process. The adversary knows some facts about the process, the ''public knowledge'' K. He is trying to deduce additional information about some ''protected'' variable *x*, based on *v*'s behaviour and the public knowledge. If he can, we say that *v* ''reads'' *x* with respect to K. Some behaviour of *x*, a priori consistent with K, is ruled out by the observed behaviour of *v*.

A similar elaboration of the nondeducibility model may allow us to limit the observers knowledge of the system. This would enable the modelling of systems with secret information incorporated into their structure, as the extra input technique does, but would not give us the ability to make any assertions about the security of that information. So, although we might be able to show that an observer who is ignorant of certain aspects of the system's construction cannot deduce other high-level information, we would not be able to easily identify the built-in secrets themselves as sensitive information and show that they are not compromised.

More attractive uses for this technique are cases where we want to assume that an observer cannot make a deduction, but where this inability is due to something other than lack of formally prohibited information. This would include, at least, the encryption example in which security depends on the extreme difficulty of inverting the encryption function.

Rather than assuming that the observer knows whether a given event sequence τ is or is not a system trace, let us suppose that there exists a filter function $\mathbf{f}$ (similar to the $\mathbf{h}$ function encountered before) that edits sequences, removing the parts that correspond to knowledge the observer does not possess. Then, given an event sequence s, the observer believes it to be a possible trace iff

$$\exists \tau \in T : \mathbf{f}(s) = \mathbf{f}(\tau)$$

Let S be the set of such sequences, which we will also call *plausible traces*. If $\mathbf{f}$ represents a restriction on knowledge, then S will be a proper superset of T.

This affects how the observer can make deductions. If we denote the secret and observable information functions as ''secret'' and ''visible'', repectively, then his *a priori* knowledge about the secret information is that it must be compatible with some plausible trace. That is, it must be secret(s) for some $s \in S$.

In a particular execution τ_0, the observer sees visible(τ_0) and therefore knows that he must be experiencing a plausible trace s with that visible behaviour, visible(s) = visible(τ_0), but in general there are many such s and he does not know which of them is the actual current trace. He may therefore deduce only that the secret information has the value

$$\text{secret}(\tau_0) \in \{\, \text{secret}(s) \mid s \in S \wedge \text{visible}(s) = \text{visible}(\tau_0) \,\}$$

For the system to be secure, there must be no extra knowledge gained for any value of τ_0. That is, for all $\tau_0 \in T$, it must be the case that

$$\{\, \text{secret}(s) \mid s \in S \,\} = \{\, \text{secret}(s) \mid s \in S \wedge \text{visible}(s) = \text{visible}(\tau_0) \,\}$$

which is equivalent to

$$\forall \tau \in T : \forall s \in S : \exists \omega \in S : \text{visible}(\omega) = \text{visible}(\tau) \wedge \text{secret}(\omega) = \text{secret}(s)$$

This is identical to the original nondeducibility requirement except for the substitutions of plausible traces S for actual traces T in the quantifications of s and ω.

This definition can be explained in English as follows: the observer has some *a priori* knowledge about what secret behaviour is possible in the system, although because his knowledge of the system itself is incomplete he considers some high-level behaviour to be possible when, in fact, it is not. The system is secure if, in every possible trace of the system, his observations are compatible (as far as he knows) with all of the presumed possible secret behaviours.

Consider the public-key encryption module diagrammed in figure 7.1. If the high-level plaintext is values from set P, the key from set K, and the ciphertext from C, we may represent a ''trace'' of this machine as a triple $\langle p,k,c \rangle$. The set of traces is, therefore,

$$T = \{\langle p_1, k_a, crypt(p_1, k_a)\rangle, \langle p_2, k_b, crypt(p_2, k_b)\rangle, \langle p_3, k_c, crypt(p_3, k_c)\rangle, \cdots\}$$

If we wish to express the notion that the observer cannot invert the encryption function, we can define the set of plausible traces as

$$S = P \times K \times C$$

which says that the observer believes that any ciphertext may correspond to any combination of plaintext and key. The security definition is trivially satisfied by this system.

This limited-knowledge nondeducibility definition not only prevents disclosure of the secret activity, it also prohibits the observer from learning anything more about the system. A security violation is considered to have occurred if the observer can, from an observation, rule out some secret behaviour that was initially believed possible, including cases where the initial belief resulted from ignorance about the system operation.

The drawbacks of this method are, firstly, that it is not intuitively easy to apply, and secondly, that it does not accurately represent the observer's knowledge about the system. These result from the fact that the essential structure of nondeducibility is unchanged, and the model still assumes that any function known by the observer can be inverted by him. Thus, in the encryptor example, when we wanted to express the idea that the observer could not invert the encryption function, it was necessary to also assume that he could not work it in the forward direction either. This is a confusing

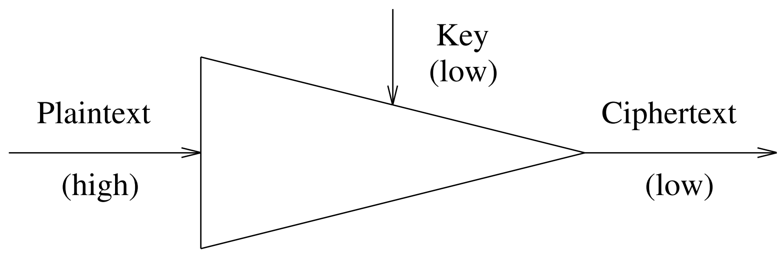


Figure 7.1: Public-Key Encryption Module.

characteristic and may make the model difficult to apply.

Alternatively, a version of the extra input technique may be useful in representing this situation. Instead of using the limited knowledge definition of nondeducibility, we may use the standard definition, and a external input may be introduced in the system definition but not included in either the visible or secret information functions. It can then be used as the source of the encryptor's output data, so that for the purposes of the model the encryptor is considered to ignore its plaintext and key inputs and simply repeat its other input. This has the same effect as the previous exercise, since the observer's ability to relate inputs and outputs to traces is reduced, but now his uncertainty about the execution trace is embedded in the external input rather than in the definition of the **f** function (and therefore the set of plausible traces S).

7.8.4. Variable Sensitivity

Identifying the sensitivity of information can be a difficult task. The conventional method of assuming that whatever is received over a high-labelled input is high is adequate if overclassification is not critical, but fails to properly model information in composed systems and breaks down entirely if downgrading is to be addressed in the model.

Nondeducibility offers unlimited discretion in the classification of information. Information functions are conveniently thought of as extracting particular bits from an event trace, but they do not necessarily have to take the form of filters. All that is necessary for the theory to work correctly is that the results of applying an information function to two executions should be equal if and only if the "information" of interest in those two executions is also equal. Whether or not this freedom can be of practical use is not yet clear, but at least in principle there is no bar to defining the "secret" and "visible" functions in any way we see fit.

The simplest test for inclusion in the "secret" function is to include all events occurring on high-level input ports. In the model of [Thomson88], where events were more complex and capable of containing both inputs and outputs, the **h** function was introduced to extract the input components from a composite event. More involved definitions may be necessary to handle downgrading.

One approach is to define a classification function that maps information onto classification levels. [Sutherland89] does this for entire events. The function he uses takes into account the context of the event, permitting an event's classification to be different in different traces or at different points in the same trace. In particular, since an execution viewed at different points in time is just a set of traces that are prefixes of one another, an event's classification can change over time or in response to other events occurring later in the execution (eg. downgrading commands).

Sutherland's scheme needs enhancement in order to properly handle downgrading, since it is not always appropriate to consider an entire event as being at a single sensitivity level. It may also be useful to distinguish between the cases of downgrading in which the sensitivity of some information is initially overestimated (eg. sanitizing) and those where it actually drops.

First, we need templates for the ''secret'' and ''visible'' information functions that will allow us to separate the different aspects of an execution that may have different sensitivities. Consider the system diagrammed in figure 7.2. In the subsystem on the left, a Confidential-level user enters data which is reviewed by another user running at Secret level. This user marks portions of the data as ''selected'', and at various times during the day an Unclassified-level operator issues a ''copy-selected'' command which dumps all of the selected data (which the U-user does not see) over to the subsystem on the right. The second subsystem has observers at all three levels. The question we wish to answer is how to characterise the sensitivity of the input to the right-hand subsystem to accurately represent which aspects of it must be hidden from each of the three observers. Initially, we will assume that the data entered is actually Confidential, and that the selector is actually using Secret-level information to make his selections.

If we consider an execution trace of the right-hand component, it will contain several input events of the form $\langle c - data \rangle$. It may be argued on intuitive grounds that the data is Confidential, since its original source was

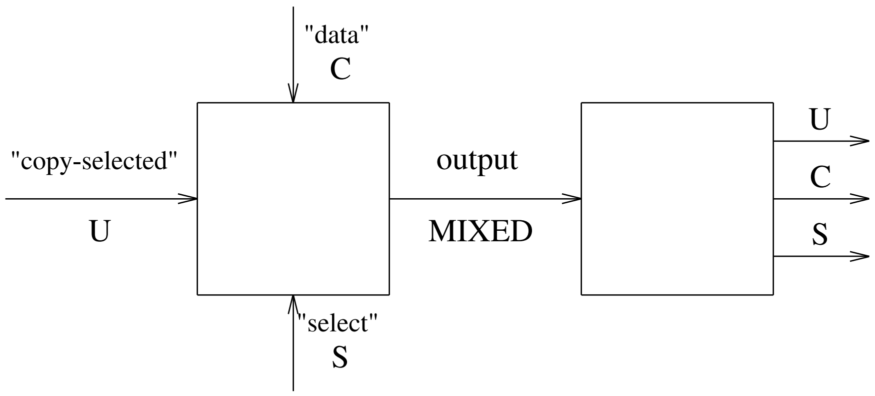


Figure 7.2: Signal with Complex Sensitivity.

that level, but it may not be immediately clear what, if any, are the Unclassified and Secret parts of the signal.

The key is to consider what kind of control is exercised over the signal by each source. The Unclassified operator triggers the input event, but does not control the contents, so it should be permissible for the U-observer to know when the inputs occur but not to know what they transmit. The Confidential input supplies data but does not select it, so the Confidential observer may see the data only if viewing it does not disclose any information about the selection process. In practice, this would require either that the Confidential observer is not permitted to see the data at all, or that an alternate path must exist so that if the C-observer does see part of the data it does not imply that it was ever selected.

Different variations may be made of this situation, for example, using an Unclassified data source, a Confidential reviewer-and-selector, and a Secret dumper. Now, the Unclassified observer may see the data but only if that will not imply that it was selected or dumped. The Confidential observer may see the data, and may see which portions were selected, but only if so doing will not indicate when or whether any dump-copies were performed. In practice, these constraints will require that if the Unclassified or Confidential observers are permitted to make any observations, those observations must be possible even in the absence of any ‘‘copy-selected’’ commands, and so there must be some other mechanism that could be responsible.

These limits on knowledge can be expressed easily and naturally if we are free to express them in terms of the inputs received by the first subsystem. If we are restricted to dealing only with the signals that pass between subsystems, the problem becomes more difficult.

Even after we have a satisfactory method of separating the various aspects of inputs so that they may be assigned different sensitivity levels, there remains the problem of selecting the correct level to assign. In the earlier examples, it was assumed that a subject operating at level X and communicating over an input port labelled level X used information of sensitivity X to form its inputs. This is the safest default assumption, since the X label guarantees only the absence of any influence from higher levels. However, in the general case these inputs may themselves be overclassified or mixed sensitivity signals, and this complicates matters further.

The first guiding principle in this assignment is the realization that the classification of a datum need only be a conservative estimate of its true sensitivity. The initial estimate may be (and probably will be) as crude as the level of the source (equivalently, the value of the input port’s label), but it is entirely reasonable to revise this estimate as further, more precise information becomes available. This further information may come in several forms, including duplication of the input by a lower level source and explicit downgrading commands issued by a reliable user.

A more accurate classification rule will permit this kind of refinement of initial classifications. The Artificial Intelligence community has long been studying the technical problem of specifying logical systems for nonmonotonic reasoning [Reiter87]. These are logical systems that permit the formulation and derivation of plausible conclusions from a combination of incomplete knowledge and reasonable assumptions, with the proviso that the conclusion may be retracted when new information becomes available. Unfortunately, these systems are complex to work with, and there remains contention among specialists as to their suitability for use at their present state of development. It may be that a suitably restricted logic, tailored to this particular application, might escape these drawbacks.

7.9. STATUS

In this chapter we described seven secrecy problems that are not adequately addressed by existing formal models. At the end of this chapter they have been dealt with in various degrees of completeness.

Invented Information
: The extra external input, *á la* quasi-determinism, appears to be both useful and appropriate. It enjoys an advantage over the Guttman-Nadel approach in that it allows the internally-generated information to be treated as having whatever sensitivity the modeller feels is appropriate.

Added Value of Computation
: A external input may also be used here to achieve the desired effect in the model, although it does not faithfully represent the functioning of the system.

Built-In Information
: Built-in sensitive data may straightforwardly be represented by external high-level inputs, or by assuming that all traces are prefixed by a standard sequence of phantom inputs that install the information in questino. Sensitive programs are best handled by recognizing that a process ‘‘reads’’ the file containing the executing program, and so that file should be treated as input data to the process. A sensitive algorithm built into the hardware might be treated by the methods described in section 7.8.3.

Filter Downgrading
: This refers to downgrading that is essentially the correction of a previous overclassification. Further development along the directions developed in section 7.8.4 should yield methods appropriate to this situation.

Time-Varying Sensitivity
: Again, development of the ideas in section 7.8.4, particularly along the lines suggested by [Sutherland89], should provide suitable modelling techniques.

Non-Invertibility

Section 7.8.3 describes the approaches that appear to best suit the inclusion of subsystems like encryptors, whose security depends on noninvertibility of some function.

Security through Unpredictability

This is the case represented by the decomposed encryptor, in which a random signal was combined with high-level data to produce a low-level encrypted output. We have no consistent and thorough model for this case. The combination of the two signals is a simple function, thus it is not reasonable to describe it as non-invertible. It is possible to treat the key stream signal as a randomizing influence, but in doing so it is removed from all information functions and we lose the ability to require that it be protected as a high-level signal. Part of the problem is that the security of the subsystem depends intimately on the characteristics of the random signal, and specifically on its statistical behaviour. We feel that it would be an excessive complication to burden a model with extensive statistical information, particularly when it is useful only in a few situations, and yet there does not appear to be any other way of preventing the hookup of just any high-level signal to the key stream input.

8 CLOSED SYSTEM SECURITY MODELLING

Before a group can enter open society it must first close ranks.
Charles Hamilton,
Black Power!,
1967.

En boca cerrada no entran moscas.
Spanish Proverb.

8.1. INTRODUCTION

In this chapter, the problem of generating composable security properties is considered to result from the attempt to define security for isolated components, and subsequently retrofit composition into the model. Such models are here called ''open'' models, in that they are based on the analysis of an individual component operating in isolation, without regard to what might actually be producing or consuming the signals that enter or exit it. This chapter suggests the use of ''closed'' modelling as a basis for the definition of security properties, where a closed model is one in which the component under scrutiny is assumed from the start to be interconnected with an assortment of information sources, sinks, and processors.

Rethinking security modelling from this perspective presents an opportunity to review some fundamental ideas about information that may profitably be changed from those conventionally used in open-system models.

8.2. INFORMATION, FLOW, AND KNOWLEDGE

Our primary goal is to express certain restrictions on the flow of information within the composite system, and to detect whether a given system design will obey those restrictions. This brings up the question of how we will define information and how we will determine where and when it flows.

The definition of information and detection of its transmission in a model is a difficult problem, and one that cannot be said to have a single, universally applicable solution. In particular, definitions that may be usefully applied in an open security model may no longer be reasonable in a closed one. Several fields of research have struggled with this question, and the many different answers that have been proposed reflect some very different assumptions. We present here a survey of some of these methods.

8.2.1. Noninterference and Nondeducibility

For completeness, we briefly mention these security definitions which have become the conventional formal definitions of confidentiality. Noninterference, which is, strictly speaking, applicable only to deterministic systems, states that a system is secure if and only if the sequence of inputs from and outputs to low-level users is unaffected by changes in high-level users's inputs.

Nondeducibility is, in practice, a similar concept, though derived from slightly different principles. It requires that the hidden information (generally, high-level inputs) not be deducible from low-level observations. Unlike noninterference, nondeducibility may be applied to non-deterministic (more correctly, probabilistic) systems, in which it takes the form of a requirement that there must exist at least one system execution that combines any given high-level input string with any given low-level behaviour.

One other difference arises if the system has any inputs that are considered neither high nor low. Noninterference requires that the low-level observations remain unaltered when high inputs change, and so these ‘‘other’’ inputs must also be held fixed. Nondeducibility, however, only requires that the high and low coexist in some execution, and so permits the other inputs to be varied to meet this condition.

Both definitions have problems dealing with probabilistic systems. Noninterference simply isn’t defined so as to be applicable to them. Nondeducibility can be applied, but may give unreasonably optimistic results if some desired system execution exists only at a very low probability.

Both definitions assume that any illicit channel may be exploited no matter how much computation may be required to recover the sensitive data. That is, they assume infinite computing resources on the part of the enemy. They also assume that the enemy has full knowledge of the principles of operation of the system, and can effectively compute all possible system behaviours.

8.2.2. Information Theory

The first quantitative treatments of information are connected with the study of communication. In connection with telegraphy, Nyquist [Nyquist24] used a logarithmic measure of the information content of a signal. If the signal can take n possible values, it can store $\log_b n$ units of information, in the sense that it would take that number of b-way signals to represent a single n-way one. The choice of b affects only the size of the units, and if the base 2 is chosen the resulting units are known as *bits* [1].

The subsequent development of communication theory led to the employment of several related measures. In considering the problem of efficiently encoding a (discrete) data stream, it is advantageous to employ a variable length code, such that high-frequency source symbols (or sequences of them) are encoded into short codes and rare source symbols into long ones. The *entropy* of a signal s^n of length n symbols is defined as

$$H_n(S) = -\sum_{s^n} p(s^n) \log_2 p(s^n)$$

where the sum is taken over all possible length n source strings. It is the average number of bits needed to encode each string s^n in the most efficient possible encoding. This code efficiency is sometimes improved, and never worsened, by encoding longer and longer strings of source symbols at a time. The theoretical lower bound on the average number of bits required per source symbol (from [Shannon49]) is

$$\begin{aligned} H(S) &= \lim_{n \to \infty} \frac{1}{n} H_n(S) \\ &= -\lim_{n \to \infty} \frac{1}{n} \sum_{s^n} p(s^n) \log_2 p(s^n) \end{aligned}$$

where the sum is taken over all s^n, the source symbol strings of length n, and where $p(s^n)$ is the probability of the string s^n. This $H(S)$, called the *entropy* (per symbol) of the source, represents its information content in the precise sense that the source may not be encoded using, on average, fewer than H bits per source symbol. If only one source symbol $s \in S$ is encoded at a time, it theoretically requires $-p(s) \log_2 p(s)$ bits in a minimum length code, from which it is sometimes concluded that the lower-probability source symbols ‘‘contain more information’’ than the higher probability ones.

In coding and communication systems it is also sometimes useful to measure the mutual information content of two signals, considered as sequences of random variables. That is, how much information from one signal can be gained by observing the other. The *average mutual information* between X and Y is defined (in [Gray90]) as

$$I(X;Y) = \sum_{x^n, y^m} p_{X,Y}(x^n, y^m) \log_2 \frac{p_{X,Y}(x^n, y^m)}{p_X(x^n) p_Y(y^m)}$$

The consistency of these definitions is demonstrated by the fact that the information $I(X;X)$ communicated by X about itself equals $H_n(X)$, the

[1] Another popular choice for b is e, yielding the natural logarithm, in which case the units are known as *nats* .

information content of X.

In deriving expressions for the capacity of noisy bandlimited communication channels, Shannon used entropy as a measure of information content of a message. His use of this measure was narrowly restricted: the entropy of a set of signals, divided by the time required to communicate them over a channel, yields the signalling rate of the channel in units that are independent of the mechanism used. Equivalently, it is the signalling rate assuming the most efficient possible encoding of the source data.

These types of measures have recently become popular in the computer security field. If the system is viewed as a communications medium, as has been done in the traditional open security model, it is not surprising that communication theory should find application. This is particularly apparent in the case of covert channels, where it may be of interest to perform a quantitative analysis of a channel's transmission capacity [Millen87]. Related probabilistic techniques have also been proposed for modelling non-deterministic systems without suffering the overoptimism exhibited by straight nondeducibility [Wittbold90] [McLean90].

However, in the study of communication systems the ''information'' source and sink are left unspecified. In fact, it is common to model the data source as being random, with some known (and hopefully exploitable) statistical properties. Whether or not there is any meaning to these bits, and whether some bits are more important than others, is of no interest to the communications engineer, and is not reflected in the information theoretic definitions.

Indeed, this lack of interest is mandated by the nature of the communication problem itself. Since those systems do not create nor interpret the bits they convey, there is no ''handle'' that the communications theorist can use to assign significance to the data. For this reason, it is perhaps unfortunate that the term ''information'' has come into popular use as a synonym for signal entropy. The bandlimited signal with the highest entropy is white Gaussian noise, which we would normally consider to be information-free [2]. On the other hand, a signal which is almost always a "0" and only very rarely a "1" (eg., only when the missile fires) has very low entropy.

Open system security models inherit a disinterest in the nature of information when communication theoretic measurements are used, unaugmented, to characterize leaky information systems. In this case, however, the disinterest cannot reasonably be justified. Information systems do often interpret data, and occasionally even invent it. The opportunity exists to

[2] It is admitted that there exist cases in which noise may have a significance, derived from its application, e.g. if it is used to generate keys or passwords. We will tentatively accept this as ''information'', and worthy of protection, although with a truly complete security model it should be possible to derive the necessity of protecting key material rather than assuming *ab initio* that it is sensitive ''information''.

distinguish between a one bit per hour leak of an unspecified sensitive file and a similar leak of a password, because the system ''knows'' in some sense which bits are passwords and which are not.

Another criticism that may be levelled at information-theoretic definitions is that they may be making unreasonable assumptions about the capabilities of the receiver or, when these methods are applied to secure systems, of the adversary. If what an observer sees is affected by sensitive information, it is assumed that the observer will be able to decode those observations into some assertion about the sensitive information. In other words, whatever ''transfer function'' is represented by the transformation of high inputs to low perturbations is assumed to be invertible.

And yet, it is known that some functions whose inversion is theoretically possible (i.e. with unbounded computing power) are, in practical terms, not invertible. For example, assume that a sensitive input consists of a set of large prime numbers, and the output observed by the adversary is their product. Since the prime factors of any number are unique, the observation logically implies the input data and so information theory concludes that the adversary possesses all the sensitive data. But, factoring large numbers is computationally intractable (for sufficiently large numbers, that is), so the function may fail to be invertible in practice.

8.2.3. Knowledge Modelling

Hintikka [Hintikka62] is reported to be the first to attempt to model the concept of knowledge, using the notion of *possible worlds*. In this approach, an agent does not know the state of the real world, but is able to rule out some possibilities based on observations. A predicate ψ is then *known* precisely if it is true in each of the remaining possible worlds. This notion forms the basis of much of the formal reasoning about knowledge in the Artificial Intelligence research community.

Logical statements about knowledge can be formally expressed and manipulated by the introduction of the *modal operator* K_i, enabling such assertions as $K_i \psi$, meaning ''player i knows ψ''. These modal operators, in turn, are often given a firm semantic interpretation through *Kripke structures*. A Kripke structure $M = (S, \pi, \mathcal{K}_1, \cdots, \mathcal{K}_n)$ is a set of *states* S, a π such that $\pi(s)$ yields, for each state s, a truth assignment to the primitive propositions, and some number of relations between states $\mathcal{K}_i$, one for each player. The modal operator semantics are then defined for K_i as follows: $M, s \models K_i \psi$ (read, '$K_i \psi$ is *satisfied* in state s of structure M'') if $M, t \models \psi$ for all t such that $(s,t) \in \mathcal{K}_i$. The $\mathcal{K}$ relations can be used to define the knowledge of each player, in the sense that $(s,t) \in \mathcal{K}_i$ iff states (i.e., worlds) s and t are indistinguishable as far as player i is concerned. One can also define a dual operator $P_i p$ as $\sim K_i \sim p$ which can be interpreted as saying that ''player i thinks that p is possible'' (i.e., does not know p to be false).

Using Kripke structures to define the modal operators means that they are, formally at least, simply new notation for conventional mathematical notions, and so inherit all the soundness and self-consistency of the underlying theory. It also means that several useful axioms may be derived for them, such as

$K_i \psi => \sim K_i \sim \psi$	''Cannot simultaneously know ψ and $\sim\psi$''
$K_i \psi => \psi$	''Can only know things that are true''
$K_i \psi => K_i K_i \psi$	''If i knows ψ, then he knows that he knows it''
$\sim K_i \psi => K_i \sim K_i \psi$	''If i does not know ψ, he knows that he does not''

Actually, the case where the $\mathcal{K}$ are equivalence relations (i.e. reflexive, transitive, and symmetric) is only one of the cases treated in [Kripke63], and is a model for a logic known as S5. In S5, all four of the above listed axioms are valid. Some work mentioned in this section are based on other logics whose models do not have equivalence relations, notably the logic S4 used in [Glasgow89], and KD45 used in [Moser89]. Figure 8.1 shows how different modal axioms are tied to properties of the model's $\mathcal{K}$ relation. (Note that axiom **T** implies axiom **D**). Various combinations of these have been given conventional names; a partial list is presented in Table 8.3. Note that some of the logics can be characterized in several equivalent ways.

Logic	$\mathcal{K}$ Properties	Valid Axioms	
		Basic	Implied
T	reflexive	**K T**	D
D	not isolated	**K D**	
B	reflexive, symmetric	**K T B**	D
S4	reflexive, transitive	**K T 4**	D
KD45	transitive, euclidean, not isolated	**K D 4 5**	
KB4	symmetric, transitive symmetric, euclidean	**K B 4**	
S5	reflexive, transitive, symmetric reflexive, euclidean not isolated, symmetric, transitive not isolated, symmetric, euclidean	**K T 5**	B D 4

Table 8.3: Some well-studied varieties of modal logics.

When the desired modality relates to knowledge, we can interpret the ''states'' as ''possible worlds'' and the relation between states as ''indistinguishability'', in which case the modal expression $K\,p$ is true in some state s iff p is true in all states indistinguishable from s, just as Hintikka proposed. Clearly, the indistinguishability relation is reflexive, transitive, and symmetric, and therefore the logic that naturally results from this formulation is the S5 logic defined above.

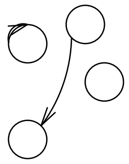
With these formalities completed, the resulting logic is a useful tool for representing and for reasoning about states of knowledge in a system. [Fagin86] contains an excellent discussion of the strengths and limits of this representation. It also describes a general model of knowledge in a multi-player system, and explores the variations that result from varying some basic assumptions about the system, such as whether it is synchronous and whether knowledge, once acquired, is ever forgotten. From this starting point logical systems have been created that represent not only knowledge but knowledge about what is known (by oneself and others) [Fagin84].

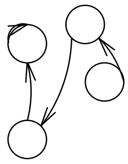
Other modal logics can also be created based on a Kripke structure model. If a temporal logic is desired, one that permits statements of the form $\Box\, p$ (read, ''henceforth p'') with the meaning ''p is true now and will remain true''. In this case, the ''states'' of the Kripke structure can be interpreted as instants in time, and the relation between them as $(s_i, s_j) \in \mathcal{K}$ iff moment s_j is at or later than s_i. This relation is clearly reflexive, transitive, but not symmetric, and so the $\Box$ operator it defines has the properties of an S4 logic. Its dual operator $\Diamond$ (read ''eventually'') then means that p is either true now or will be true at some future time.

It is also possible to create a modal logic without reference to a defining structure. For example, we may have a notion of ''belief'' that is similar to knowledge, but differs in that something we believe may turn out to be false. A logic could be designed by starting with an S5-type knowledge logic, in which all of **K**, **D**, **T**, **4**, **B**, and **5** are valid, renaming the modal operator B for belief, and removing the axiom **T:** $B\,p => p$ so that belief no longer implies truth. However, without the Kripke structure as a guide it may not be apparent that the original set were not independent. In fact, **T** is a consequence of **B**, **4**, and **D**, or alternately of **B**, **4**, and **5**. To thoroughly eliminate **T** we must remove both it and **B**, which yields the self-consistent logic KD45. Its Kripke structure has the not isolated, transitive, and euclidean properties, but it is not immediately clear how to interpret the states of that structure in a meaningful way[3].

---

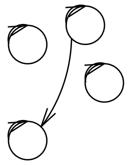
[3] The following is a possible interpretation: In addition to the groupings of indistinguishable worlds introduced in the knowledge logic, the (possibly incorrect) beliefs we hold about how the universe operates also partitions the possible worlds into ''ideal'' (or ''believable'') and ''nonideal'' worlds, such that in any group of indistinguishable worlds at least one of them is ideal. Within each group, there is an arrow from every world to every ideal world. An arrow from a to b means that if we are in world a, we may believe that we are in world b, because it is both believable and indistinguishable from a. Nonideal worlds do not conform to our beliefs, so are not the destination of any arrow.

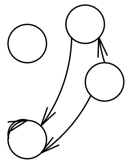




General relation　　Non-isolated (arrow leaving each state)

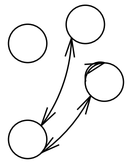
axiom **K:** $K p \wedge K (p => q) => K q$　　add axiom **D:** $K p => \sim K \sim p$

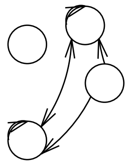




Reflexive　　Transitive

add axiom **T:** $K p => p$　　add axiom **4:** $K p => K K p$





Symmetric　　Euclidean ($\alpha \mathcal{K} \beta$ and $\alpha \mathcal{K} \gamma$ imply $\beta \mathcal{K} \gamma$)

add axiom **B:** $\sim p => K \sim K p$　　add axiom **5:** $\sim K p => K \sim K p$

Figure 8.1: Kripke Structure Properties and Corresponding Modal Axioms.

Within the security research community, specific modal logics have been used to reason about authentication protocols [Burrows88]. Various logics with multiple modal operators have also been proposed:

- knowledge and permission [Glasgow88] to deal with confidentiality issues
- knowledge, obligation, and time [Glasgow89] which also allows the expression of some form of integrity constraints
- knowledge, permission, and time [Cuppens90] to handle confidentiality concerns in a dynamic environment

- knowledge and belief [Moser89], to describe security-related reasoning that is based on presumed or probable truth rather than certainty.

The last of these permits the formal expression of assumptions that would otherwise be implicit in the use of the model, for example the belief that "only player X knows X's password". The paper also extends the belief portion of the logic to form a non-monotonic belief logic, which allows for assumptions to be refuted by subsequent contrary evidence [4].

Addition of these modalities to a logic yields the basic tools sufficient for representing states of knowledge, permission, or belief, as the case may be. They can also be used to express prohibitions on knowledge, thereby describing confidentiality requirements, or obligations of knowledge, which encompasses some portion of an integrity specification. The formal specification of security policies is a prime area for the application of modal logics.

The knowledge logics may also be used to model a player's reasoning, based on the axiom

$$K_i\, p \wedge K_i(p \rightarrow q) => K_i\, q$$

which says "if i knows p, and also knows that p implies q, then i knows (i.e., can deduce) q". Most logics of knowledge also have an inference rule of the form

$$\text{if } \psi \text{ is a theorem, then } K_i\, \psi \text{ is a theorem}$$

which implies that all players know the principles of operation of the system, at least insofar as these are reflected in the theorems of the logic, but if this rule is eliminated then a logic can be created in which players are ignorant about some system specifications.

However, there are reasons to temper our enthusiasm about modal logics. It is not difficult to create incomplete logics, in which valid statements cannot be proven. For example, [Glasgow88] describes an S4-based knowledge logic, but the model presented for the logic is a Kripke structure whose $\mathcal{K}_i$ are equivalence relations. In any such model, all the axioms of S5 are valid, and in particular the so-called ''negative introspection'' axiom (**5:** $\sim K_i\, \psi => K_i \sim K_i\, \psi$) is valid, but not provable in the S4-type logic. This is not necessarily catastrophic, but it does mean that some truths, possibly ones we are very interested in, cannot be proven in the logic.

We may also criticize the previously suggested use of knowledge logic to model a player's reasoning. Any such attempt must be approached carefully, as it amounts to an assumption that the opponent is capable only of the kinds of simple deductive reasoning captured in the logic. This is almost certain to

---

[4] These ''default reasoning'' logics are difficult to employ and their properties are not as well understood as are those of more conventional monotonic logics.

underestimate some of an opponent's capabilities, since it does not account for inferential, intuitive, or other non-deductive kinds of reasoning that an opponent may employ. At the same time, it may overestimate the player's capabilities in other areas, because it assumes that the player's reasoning powers are not computationally limited.

Although modal logics yield tools to represent and manipulate the epistemic (i.e., knowledge-related) consequences of system activity, they do not give any guidance in answering the question of how knowledge is acquired as a result of that system activity. Only after the system functional specifications have been converted into statements in the logic, representing the primitive epistemic truths that result from user's interactions with the system, can the logic be used to determine their consequences.

Along with this limitation, the logics likewise do not provide a means of handling the dynamic interactions occurring within composite systems. On the other hand, the logics do provide a direct and common representation of security policies and their operation, and so open the door to the analysis of the composition of policies.

This suggests that a system of modal logics could be a valuable complement to an operational model of a composite system. First, the operational model would be used to capture the interaction and interdependence of component function. Results from this level of modelling must then be translated into system-specific modal axioms in the logic. These may be combined with and checked for consistency with axioms describing the security policies of the components and the overall security objectives of the system. An unanswered question here is how much of this work may be performed ahead of time for a large class of systems. Since some part of the logic is created only after the component functional composition step is complete, it may be difficult to get significant results without considering the details of individual systems.

8.2.4. Zero-Knowledge Proof Systems

Computational complexity is a name given to the study of the innate difficulty of solving computational problems, the classification of otherwise dissimilar problems into groups according to measures of this difficulty, and the investigation of fundamental differences between the resulting groups. Early work in the area was concerned with providing a basis for the comparison of algorithms, and this quickly led to the question of optimal algorithms and proofs of optimality.

In the application of complexity theory to security, the importance of determining optimal algorithms has been eclipsed by that of establishing lower bounds for the difficulty of problems so that they may be considered practically unsolvable by an enemy. The fundamental tenet of complexity-based security recognizes that an opponent's computational resources are not limitless, and that this fact can be exploited to yield effective security in ways that more demanding definitions would not condone.

The theory assumes that, for any problem, the possible methods of solving it can be ranked according to some agreed cost measure, such as time to compute or storage required. It is also usually expected that the cost of solving a particular instance of a problem can be related to some simple measure of the problem's size, such as the number of symbols (from a finite alphabet) required to describe the specific instance of the problem.

Many different costs can be arrived at for a given problem. The choice of measuring time rather than space obviously yields different measures. Different measures are also obtained when maximum, minimum, or average cost over a particular class of problems is considered. Less obviously, the definition of what it means to ''solve'' the problem also affects the result. There exist efficient algorithms that give approximate answers, where exact ones are prohibitively expensive. Related to these are algorithms that give the correct answer exactly most of the time, but the wrong answer some of the time. A now classic problem that can be efficiently ''solved'' in this sense is determining whether a given number is or is not prime. Straightforward solutions require time that is exponential in the length (i.e., number of digits) of the number, but an answer that is usually right can be computed in polynomial time [Solovay77].

Measures can also be distinguished according to the model of computation assumed. Deterministic Turing machines define one standard of computation. Probabilistic machines add the ability to flip coins during the computation, which is necessary to implement algorithms such as the primality detector of the previous paragraph. Nondeterministic Turing machines define an unrealizable level of computational power: like the probabilistic machines, each state has more than one possible successor, but the nondeterministic machine simultaneously follows all possibilities (equivalently, it consistently makes the most advantageous choice).

Complexity considerations, formal or otherwise, have always played a part in security, particularly in cryptography. More recently, the application of complexity-theoretic concepts to interactive proof systems, and particularly to ''zero-knowledge'' proofs, has resulting in new and interesting definitions of information and information flow. The following description follows [Goldwasser85].

An *interactive proof system* is a pair of Turing machines (specifically, probabilistic Turing machines) that share an input tape and also share a pair of communication tapes. Both machines can only read from the input tape. One communication tape is write-only for one machine and read-only for the other, while the second communication tape operates in the other direction. The machines take turns computing, sending a message to the other machine, and waiting for a response. Either machine may end the computation by halting during its turn.

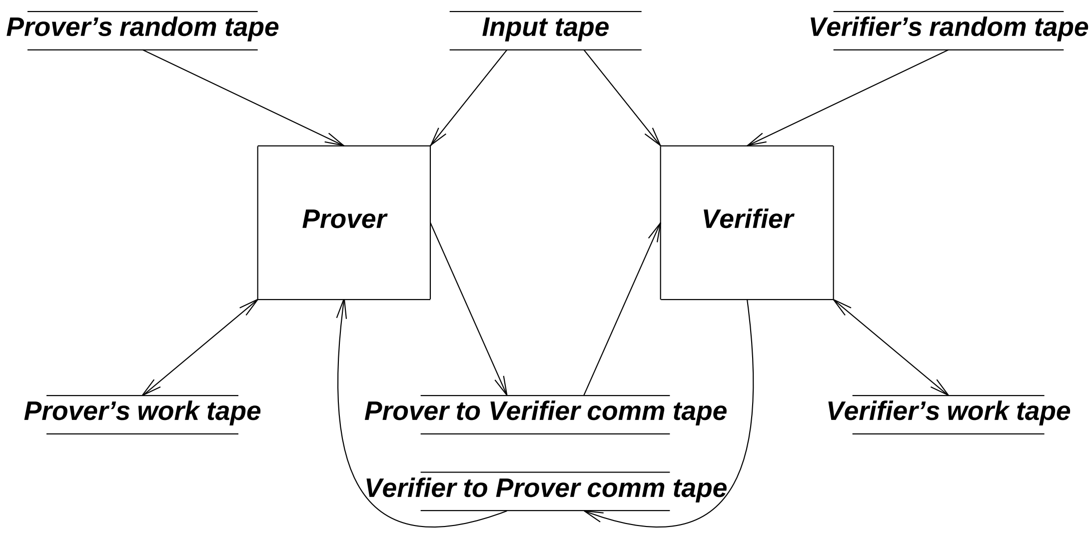


Figure 8.2: An Interacting Pair of Turing Machines.

One machine, the *prover*, is assumed to have infinite computing power, while the *verifier* is limited to polynomial time computations. Given a language (set of strings) L, and a tolerance k, these machines form an interactive proof system for L if, when presented with an input string x of length n, they have the properties:

- if $x \in L$, then the verifier will halt indicating acceptance with probability at least $1 - \frac{1}{n^k}$ for sufficiently large n.
- if $x \notin L$, the verifier will accept with probability at most $\frac{1}{n^k}$ for sufficiently large n, even if the prover is replaced with an arbitrary machine *prover*′.

The ''fact'' that is being proven here is *the input x is a member of language L*. The first condition says that if the statement is true, the verifier will almost certainly be convinced. On the other hand, the second condition requires that if the statement is false, the verifier is unlikely to be fooled, even if a dishonest prover is substituted. This means that the verifier does not have to trust the prover, but has the means to check (in polynomial time) the credibility of what it is being told.

Interactive proof systems for languages can be trivially constructed if the languages are sufficiently easy to recognize. For example, if the language is the set of all prime numbers, the verifier can use the Solovay-Strassen technique to check the input for membership, with small probability of error, in probabilistic polynomial time, without any interaction with the prover.

Nontrivial cases arise when the task requires the prover's greater computational power. Then, the verifier must accept some information from the prover, but must do this in such a way that it can check the information for reasonableness. Typically, this takes the form of a sequence of challenges that the verifier puts to the prover, and to which the prover responds.

Of course, the responses from the prover may tell the verifier more than simply whether or not the input is in the language under consideration. If prime testing were a harder problem, a prover might demonstrate that a number is composite by replying with one of its factors, but that reply would communicate more than the desired fact. In particular, it is only necessary to communicate that a factor exists, but this protocol actually gives its identity. What distinguishes the zero-knowledge work is that this extra information, transmitted by the prover, is quantified and limited.

In discussions of zero knowledge, ''knowledge'' is taken to be new knowledge, or information received that the recipient did not already possess. In turn, ''possession'' needs to be precisely defined, preferably in a way that depends only on observable characteristics of the possessor. The definition chosen is that a machine knows any thing that it is capable of outputting. Since, in the model of figure 8.2, the input tape is available to both machines, a lack of knowledge implies an inability to compute some value rather than a lack of inputs.

A consequence of this definition is that whether transmitted information constitutes knowledge depends on the computational capabilities of the recipient. Learning a random string is knowledge only if the learner is not capable of generating its own random strings with similar statistical properties. Learning the factors of a number is knowledge if the recipient has polynomially bounded resources, but not if its resources are unbounded (in that case, it could have factored the original number itself).

The theory also carefully defines what is meant by the phrases ''capable of outputting'' and ''similar statistical properties''. Essentially, the verifier is considered to gain no (excess) knowledge from its conversation with the prover if it is possible for a machine with computing power equivalent to that of the verifier (that is, probabilistic polynomial time) to generate plausible conversations. Plausibility, too, has a complexity-based definition, or rather three definitions that yield three variants of zero-knowledge. Considering the collection of all authentic prover-verifier conversations, and the collection of all synthesized conversations, if there exists no probabilistic polynomial-time *distinguisher* that can tell them apart with any significant reliability, then the system is a *computational zero knowledge* system. If the collections appear equal to any distinguisher with unbounded computational power, this represents *statistical* zero knowledge. Finally, if the two collections are actually equal, the system is said to be a *perfect* zero knowledge system.

Zero knowledge proof systems are a very specific type of secure communication. Attempts to apply the theory to secure systems in a general way would probably not be very fruitful, so the intricacies of its definitions are not really germane to our further considerations and we will not go into any greater detail about them.

What is interesting, though, is the distinctive treatment of knowledge and information flow that the theory employs. Contrast the view expressed by the complexity based definitions with the way these concepts were managed in the theories described earlier. Those theories assume some external source of sensitive information that is visible only to a subset of the entities in the system. They also assume that an observer can determine all the logical consequences of its observations. As mentioned in a previous chapter, this last assumption makes it difficult for those theories to deal with sensitive information that is generated by processing nonsensitive sources, because the observer is assumed able to duplicate the processing.

The present theory, on the other hand, takes a rather different approach. Inputs to the composite prover+verifier system are public information. One participant, the prover, is capable of performing computations that the other participant cannot duplicate. The secrets in this system are exactly the results of these computations, performed on the nonsensitive public inputs. That is, the only kind of secret information here is that produced by the added value of processing, which is the kind that the more conventional security definitions cannot handle. Conversely, secrecy resulting from the private nature of inputs is not handled here, though it is by the conventional methods.

This theory also recognizes the existence of computational limitations, while other theories did not. These limits exist not only in the verifier, but also in the definitions of computational and statistical zero knowledge. The ''knowledge content'' of the conversations (relative to the verifier) is defined in an empirical way, as the amount of information that a computationally limited observer (the distinguisher) would be able to extract [5] by comparing the real conversations to simulations created by something like the verifier.

There are several advantages to adopting complexity-based definitions for security. Removal of the assumption of the adversary's unbounded computing power is a significant improvement. Information-based formal security definitions are often criticized as being too ''strong'', in that they tend to prohibit for theoretical reasons some system behaviours which are, in

---

[5] This is not apparent in the abbreviated definition given here, but the full definition in [Goldwasser85] computes the number of bits of information that the distinguisher can extract, and defines zero knowledge as zero bits of available information.

practice, quite adequately secure. There is the danger of underestimating the opponent's capabilities, but there are models of computing available (such as probabilistic polynomial time computation) that include computations that are approximate, involve guessing, or sometimes give a wrong answer, and these seem to be a pretty comprehensive characterisation of what we might intuitively call useful methods of computing.

Drawbacks of the complexity approach begin with the fact that using complexity-based definitions significantly complicates the process of evaluating a model. Proving computational properties of specific problems or algorithms is difficult enough, but proving them for a sizable, incompletely specified system is likely impossible. This difficulty exists even in the zero-knowledge research itself, where the non-absolute definitions of knowledge (that is, computational and statistical zero knowledge) are encountered less often than the ''perfect'' variety [6], because the latter is easier to work with, even though it is probably excessively strong.

This difficulty becomes even more likely if the system is not designed to have any particular complexity properties. Thus, allowing interference from high to low behaviours provided that no knowledge can be extracted is a reasonable policy in principle, but such a constraint is really a requirement that the high-low interference path have cryptographic properties of some known strength. Any interference of this kind is usually a manifestation of undesired but unavoidable interactions within the system, and is not purposely designed in, and so it is not reasonable to expect it to be, coincidentally, a respectable encryption. In order to apply complexity-based methods, it would probably be necessary to design in some tailored ''noise'' in addition to the naturally occurring interference.

A second concern relates to what it means for a problem to be a member of a particular complexity class. It is a carryover from the origins of complexity theory that a problem is typically characterized by the maximum difficulty over all instances of the problem. The statement that ''factoring large numbers is difficult'' usually means that there exist some large numbers that are difficult to factor, but does not rule out the possibility that some large numbers are easy. Clearly, the 200 digit number consisting of a ''1''followed by 199 ''0'''s falls into the latter category. When using computationally difficult problems to provide security, it is necessary to avoid accidentally choosing such an easy instance. The definitions must be changed to consider the minimum rather than maximum difficulty of all problems in a class, so as to yield a guaranteed resistance to compromise. In the case of factoring, this would require that any proposed large number meet some additional stipulations before being considered for use. That is, rather than being content with the conventional computational complexity statement that ''some large

[6] [Bellare90] remarks that ''we do not have too many examples of languages with statistical zero knowledge proofs''.

numbers are difficult to factor'', we need a test T that enables us to say that ``all large numbers n satisfying $T(n)$ are difficult to factor''.

8.3. COMPOSABLE FORMAL SECURITY MODELLING

Intuitively, a system is secure if high information does not leak to subjects lacking sufficient authorization to possess it. This holds true whether the system in question is a simple or composite one. Assuming separation of the wires that connect the components of a complex system, and the observation of rudimentary and obvious rules about what kind of outputs connect to what kind of inputs, this leakage (or ``crosstalk'') if it occurs at all must occur within some component of the system. From these observations, we may consider an component *simply secure* if it does not permit leakage when used in isolation, and *composably secure* if it not only does not leak, but cannot be induced to do so however it is interconnected to other similarly qualified components, given that appropriate and simple constraints are observed on the manner of hookup.

The foregoing rationale underlies previous approaches to the composition of security. A similar view of the problem (omitting the parts about composition) is evident in earlier work defining fundamental formal security properties, such as noninterference [Goguen82] and nondeducibility [Sutherland86]. In each case, the system of interest (simple or composite) is considered to operate as if in a vacuum, with input signals appearing on its input ports and output signals disappearing into the void. As the inputs are assumed to take on all possible values, the outputs are monitored to verify that there is no leakage of high-level information into the low-level signals.

These are *open* models, in that they define security in terms of labelled signals crossing the boundary of the system of interest (which will be designated the SOI), without regard to what might actually be producing or consuming those signals. In the days before the raising of the composability issue, there was also an unstated assumption that a component which was secure in isolation would naturally remain leak-free in composition.

McCullough [McCullough87] challenged this assumption when he discovered that components that were secure, according to a formal definition of the kind that was reasonably popular at the time, could when interconnected coerce one another into leaking high information to low outputs. In the formal definition he used, ``secure operation'' means that the range of behaviours visible to low-level observers must not depend on high-level inputs to the system. That is, all possible observations of low-level input and output sequences must remain possible if high inputs are added or deleted from the trace. This definition does permit the high inputs to perturb low-level activity, but only if these perturbations are roughly the same "shape" as naturally occurring variations, so that observers cannot distinguish leaks from noise.

It is possible to take components that meet this security criterion and interconnect them forming a composite machine that does not. A typical scenario in which this happens has at least one component that is capable of secure operation, but only if some condition is met with respect to its high-level inputs and outputs. The condition referred to is something achievable, such as ''*the number of high outputs produced equals the number of high inputs received*'',and the component strives to achieve the condition. It sometimes fails, but the unreliability of that failure is considered by the model to render the leak sufficiently noisy — basically, the low-level observer has no means to detect which perturbations are leaks and which are normal random variations.

In figure 8.3 some execution traces are shown for a component that fits this description. Progession of time is represented by moving upward in the trace diagrams. The first trace shows low-level activity in the absence of

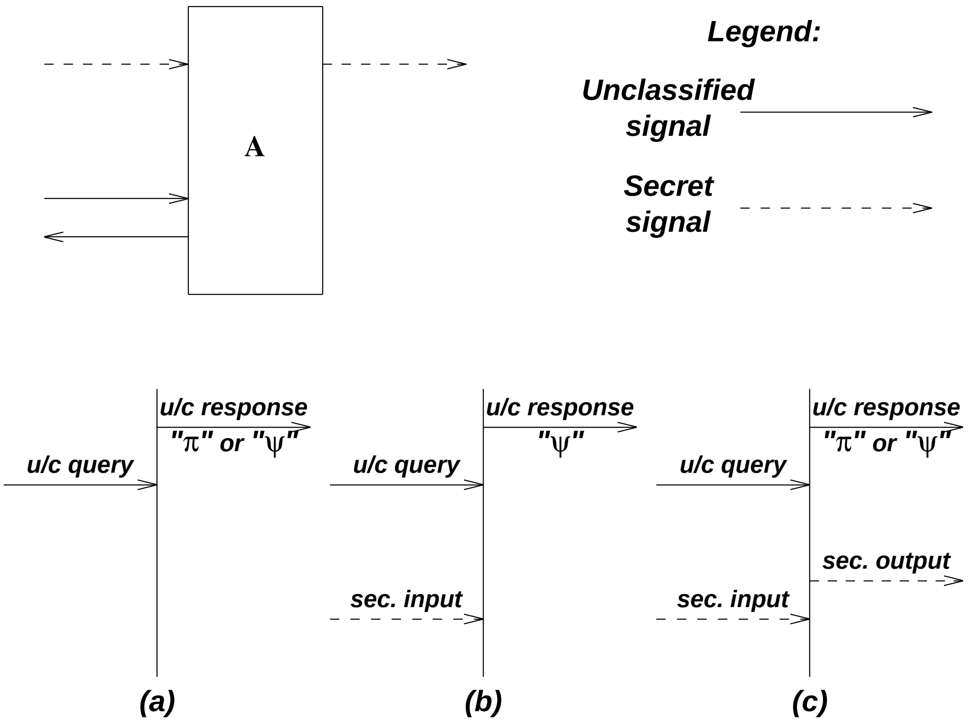


Figure 8.3: A ''Secure'' Machine and Some of its Possible Execution Traces.

high-level messages. A low query results in a low response that may take either of two values, π or ψ. If a high-level message is received, the possible low-level output values depend on whether the system is able to relieve itself of that high-level message before the low output is produced. In trace (b), the low output takes only the value ψ, while in (c) the component has managed to unburden itself by producing a high-level output and can again produce either of the two low-level responses.

This component is secure according to the definition given earlier. This is so because both low responses (π and ψ) remain possible no matter what high *inputs* are received. The fact that the timing of high *outputs* makes a difference is not a consideration in this model. Although this system seems artificial, it might correspond to a realistic system in the following way: the component is actually a dual-processor computer, and in its low-level response it indicates which processor executed the query. One processor, however, may be pre-empted to handle Secret message traffic, and so if there is ongoing Secret processing when the low query is answered it can only have run on the second processor.

The difficulty with composing this component is that it may be connected such that its inputs are no longer independent of its outputs. Consider connecting two of these units as shown in figure 8.4, with the high outputs from each as high inputs to the other [7]. In trace (a), there are no high inputs and so both systems are able to produce either low response. In particular, it may happen that both systems produce the π response. A single external high input to the left component will violate its condition, and if it does not remedy that situation it will not be able to produce the π output, as shown in trace (b). On the other hand, if it does produce a high output, that output will form an input to the component on the right, and will cause a violation of its high-signal condition. The two systems may then exchange this burden many times, as in trace (c), but one or the other of them will not be able to produce the π response. The addition of a high input to the composite system has eliminated one possible low-level behaviour, so the composite machine is not secure by the original criteria.

McCullough's response to this, since emulated by others, was to search for and propose a fixed-up formal definition of confidentiality that prevented these kinds of destructive dynamic interactions. However, it remained the case that the composite system itself was viewed and analyzed in an open context.

Another possible reaction, and one that might, on reflection, seem more natural, is to attribute noncomposability to a failure of the open system modelling approach. The entire concept of the ''security'' of a device only has

[7] McCullough does not use examples where signals merge or fan out, so his examples are somewhat different from this one, but the principle is the same.

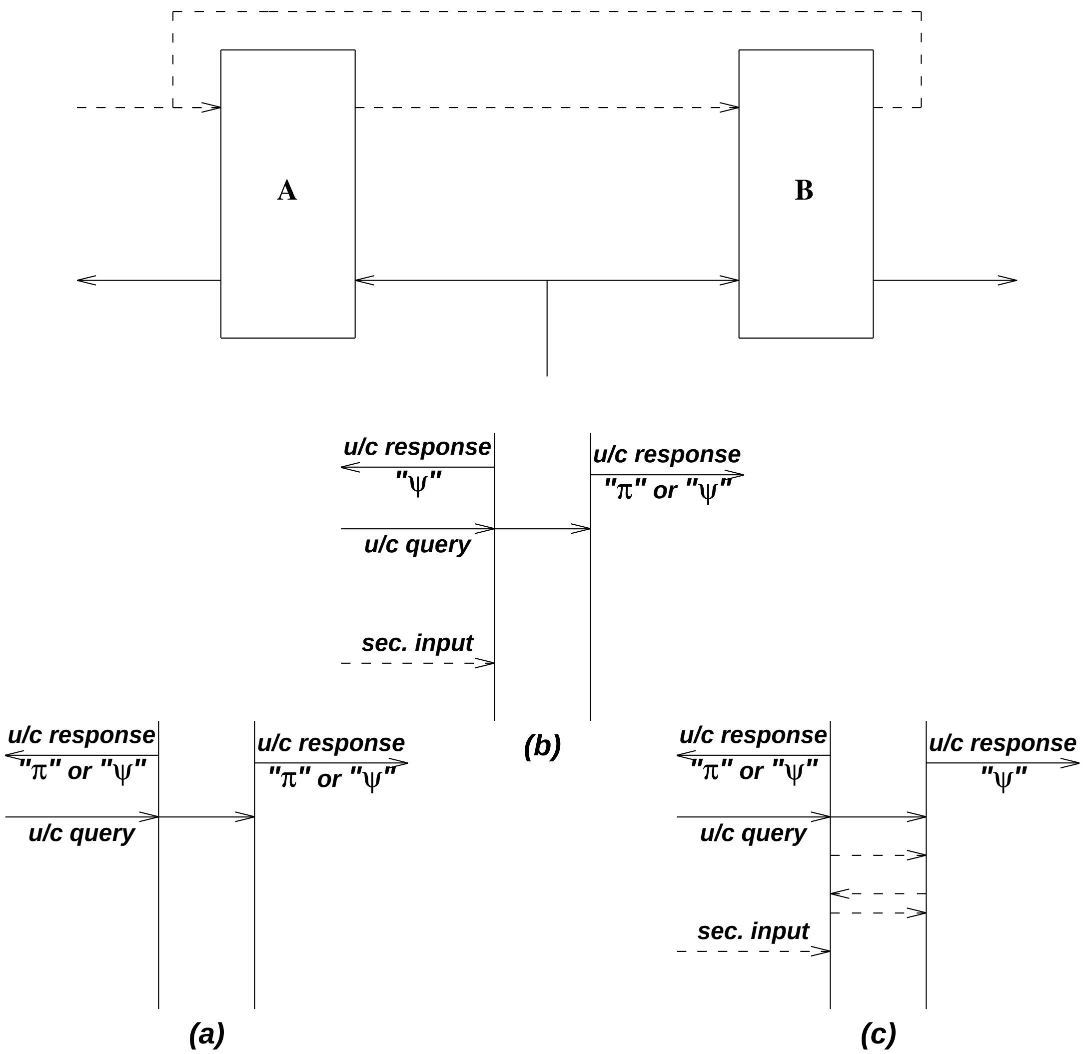


Figure 8.4: An Insecure Composite Machine.

meaning when that device is connected to something, because security relates to how a device functions when connected to information sources and sinks, to other devices, and to users. Any device is secure as long as its ‘‘ports’’ are left unconnected, so the distinction between secure and non-secure devices cannot be made in a natural way when devices are treated in isolation. In their attempt to get meaningful results from open models, researchers were forced to devise formal definitions of security defined in an open framework. The open models assume that signals enter the SOI from some unspecified ''beyond'', a beyond which, it was acknowledged, might be malicious, but which was not expected to possess any guile. In particular, it was implicitly assumed that an ''attack'' consisted of preselecting a sequence of inputs, presenting them to the SOI, and watching the outputs that resulted; this assumption entirely discounted the notion that an opponent might choose which inputs to submit in real time, based on previously received outputs. Presumably, the resulting definitions were expected to lead to secure composite systems, but with only intuition as a guide the initial attempts were flawed, as McCullough showed.

From these observations alone it is reasonable to require that modellers using open models provide some convincing argument that the results of their analysis of isolated components can be usefully applied to real configurations. Alternatively, nondisclosure can be approached using a **closed system** model in which entities external to the SOI are explicitly represented, with as much or as little detail as is implied by the known constraints on their behaviour, and by our willingness to enforce further restraints.

8.4. CLOSED-SYSTEM SECURITY MODELLING

In a closed-system type of model, we might expect that we take as our starting point the premise that the system of interest will be interconnected with other components. Any functional security objectives we may have should properly be expressed as properties of the resulting composite system. It may then be possible to derive acceptable componentwise properties for the SOI, if it can be shown that every system that can result from its composition, with prescribed other components according to permitted interconnection rules, will display the desired closed-system secure behaviour.

In the following sections we will develop a modelling framework using a closed system approach. An attempt will be made to describe the model in general terms, delaying selection of particular definitions or interpretations as long as is practical. The objective in doing this is to emphasize that closed-system modelling is not a particular model but a framework for developing models that attempts to avoid the surprises that seem to result from using the open-system approach.

For the sake of simplicity, let us first assume that our System of Interest is a multilevel system dealing with two levels of information, **High** and **Low** . It is connected together with a number of other entities, some of which are also multilevel systems, and others which are single-level High or Low.

Automata theory teaches that any finite aggregate of reasonable computing engines may be simulated by a single unit of the same type. For example, any finite collection of N Turing machines is equivalent to a single N-tape

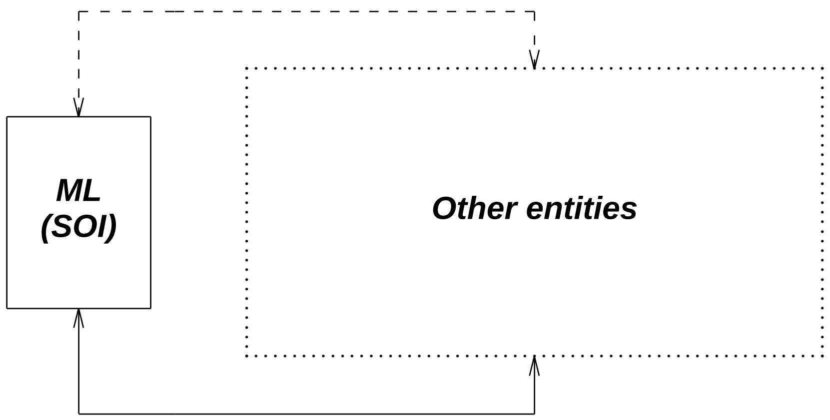


Figure 8.5: Multilevel System of Interest in a Closed Environment.

machine [8], which is computationally equivalent to a single 1-tape Turing machine [9]. This may be significant from two perspectives: we are interested in characterizing the possible behaviours of the aggregate of non-SOI components, but we may also be interested in the computational capabilities of the adversary if security in our model explicitly depends on the intractability of some problem.

This equivalence permits us to gather together the non-SOI components into single similar units, and so reduce the arbitrary system of figure 8.5 to that of figure 8.6, where the SOI has been kept separate but all other components have been absorbed into representative multilevel, high level, and low level equivalents. In the figure, solid lines represent connections that would be labelled ‘‘Low’’,and dashed lines connections labelled ‘‘High’’. The degree to which those labels reflect the actual sensitivity of the information on the connections is a question of policy and of the level of policy compliance in the composite system. The former is something that must be specified, while the latter is what we want our model to tell us. The only remaining feature of note is the one-way connection from the Low aggregate to the High, as its classification changes from one end to the other. This anticipates a policy that permits write-up, and explicitly demonstrates that the Low entities may communicate to the High (for whatever purpose, including collusion) without the awareness of the SOI.

[8] Strictly speaking, this requires that the original N machines operate synchronously.

[9] See [Hopcroft69] or any standard reference on automata theory.

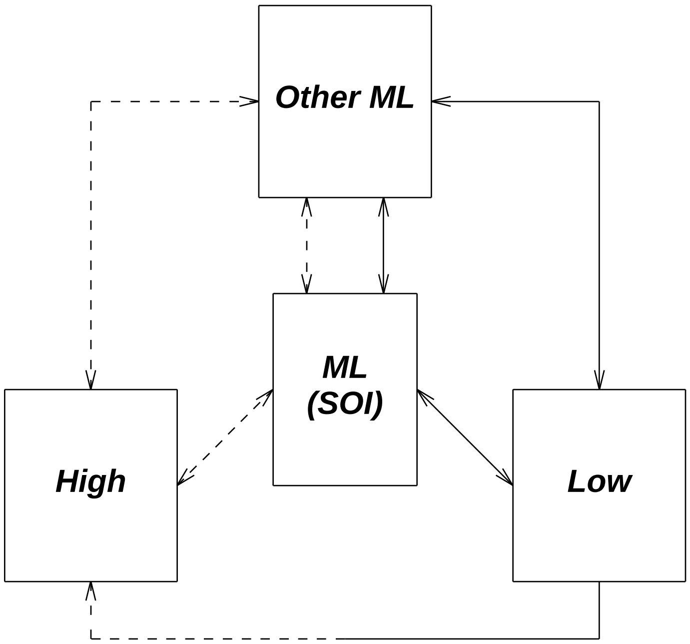


Figure 8.6: Simplified Closed Model of Multilevel Operation.

Whatever is chosen as the objective policy with relation to the flow of information, we will add a requirement similar to the following:

> *The SOI security requirements must be such that the system security objective is met for all permitted composite ''High'', ''Low'', and ''Other ML'' systems.*

Further progress requires that we settle some basic issues related to defining our security objective, and also that we establish some principles, or ''ground rules'', that outline the kinds of restrictions that we expect to hold or will be willing to enforce outside the SOI. It would, for example, be impossible for the SOI to enforce security if the other ML systems do not, or if the High and Low systems are able to establish new illicit communications bypassing the SOI.

With regard to the other ML systems, it is sensible that all multilevel components in the system should be subject to the same set of rules, so a reasonable meta-policy to follow would be that whatever criteria is developed for the SOI must also be adhered to by the other multilevel entities. However, our desire to aggregate these entities in our model adds another wrinkle to the problem.

Some number of multilevel components are aggregated into something that is similar in its computational capabilities, and clearly is also multilevel, but we have no assurance at this point that the aggregate's behaviour will satisfy the SOI criteria just because its components do. To guarantee that, we must further require that the SOI criteria be composable. The difference between requiring composability here and requiring it in the open system model is that, in the open system, composability was taken to be an end in itself, while here there is the clear technical motivation that it helps obviate the consideration of an infinite number of system configurations.

An effective meta-policy for the single-level components requires somewhat more reflection. They may represent equipment or software which, since it is not operating in multilevel mode, may not be subject to the same sorts of design, manufacturing, and acceptance procedures that are applied to the ML components. We may not be justified in placing any significant reliance on our knowledge of how these components behave.

Single-level components may also represent people who, though they may be trusted not to intentionally contravene security, might inadvertently contribute to a violation engineered by some other component. In any case, it is doubtful whether any available techniques would be adequate to describe any useful set of assumptions about the behaviour of users.

Instead, the only assumption we will make about the single-level entities is that they are restricted to the communication paths depicted in the configuration of figure 8.6. In the case of untrusted entities, this restriction is externally enforced. In the case of trusted users, we assume their trustworthiness precludes their entering into new, unauthorized communications. Beyond this, however, we will place no restrictions on the behaviour of these entities.

8.4.1. Derived Component Security

Having made a case for closed-system modelling in a confidentiality setting, we may now take one step backward and consider the closed-system approach as applied to general security properties.

In this section we will, in very general terms, construct a closed-system security model that will allow us to relate system security objectives to component specifications. We begin by introducing the pieces of the model:

Given:

S	a predicate on sets of executions, defining a system security objective
T	an indexed set of predicates t_1 through t_N on sets of executions, specifying N components

t_0 a distinguished predicate on sets of executions, specifying a distinguished component (the component of interest).

D a set of collections $d = <d_0, d_1, \cdots, d_N>$ of mappings from (system) executions to (component) executions

For every system execution e, $d_i(e)$ is the execution of the ith component when the system is composed according to $d = <d_0 \cdots d_N>$. We will use the shorthand $d_i(E)$ to denote the image of E under d_i, that is, the set $d_i(e) \mid e \in E$.

Then, given the definitions above, specification t_0 is *composably S-secure for D with* T iff, for every $d \in D$, every set of system executions E satisfying

$$t_0(d_0(E)) \wedge t_1(d_1(E)) \wedge \cdots \wedge t_N(d_N(E))$$

also satisfies predicate S.

That is, any set of system executions that can be decomposed, using a consistent set of permitted (de)composition functions, into valid component executions, must satisfy the system security objective. It is phrased in terms of decomposition rather than composition simply for ease of expression. The sense of it is that any component implemented in accordance with the t_0 specification, and composed in any of the prescribed ways with components meeting the t_i specifications, must result in a composite system that is secure in the sense implied by S.

In this way, we expect to take a security objective that is expressed in closed form, and to relate it to a corresponding open-style security specification t_0 that can be applied to isolated components. In consequence, the precise nature and limits to the security delivered by the t_0 specification will be apparent in the closed-system security property S and in the explicit characterization of the other system components (T) and of the nature of the composition (D).

We will now consider the pieces of this definition in more detail.

8.4.1.1. Executions

In the preceding section we have not defined precisely how ‘‘executions’’ are represented, because there are many approaches with varying levels of expressivity, manipulability, derivability, or different sets of underlying assumptions about how components can interact. There are some implications about the level of detail that the system execution must include, which will be described where they arise.

The different methods available to represent collections of executions include direct methods such as event traces and state traces. Events are atomic actions, and an event trace is a sequence of such actions, possibly annotated with timing information. States are assignments of values to objects, and their traces record executions as successive state changes.

In some sense, the two methods are equivalent, since events resemble state transitions and states are memories of past events. The equivalence is not simple nor exact, however, because an event is more like a mapping from current to next state (i.e., a collection of transitions depending on the current state), and states are usually equivalence classes of event histories reflecting the fact that a state can usually be reached in more than one way. State traces also have the notion of different initial states, which has no direct analogue in the event trace formalism.

Executions may also be represented indirectly, by describing some type of structure that can be operated on to produce the desired set of event or state sequences. State machines may be used in this way, and it is often convenient to use a simple state machine in a model if the modelled system is a more complex state machine. The semantics of the CSP notation (Communicating Sequential Processes, [Hoare85]) may be described by event traces, and therefore the notation may be thought of as an algebra whose operations compose traces.

8.4.1.2. Security Properties

The predicate S is defined on sets of executions, each set being interpreted as the system behaviours achievable by a particular component set composed in a particular way. This means that the security definition may include statistical requirements on the distribution of executions, restrictions on correlations between executions, or liveness conditions, that would not be expressible if the definition were to consider only one execution at a time.

Broadening the scope of the definition in this way means that we can express, in the closed system format, security properties that are similar to the noninterference and nondeducibility properties used in open system modelling. The common element between them is that they permit a given execution depending on whether or not some related execution (eg., one with all high inputs purged but with the same low activity) can also be observed. These are apparently useful forms for security properties to take, although they may be modified somewhat for use in the closed system context. They are also properties whose satisfaction cannot be decided by inspecting executions one at a time.

8.4.1.3. Specifications

The various predicates t_i represent specifications of components. A component meets its specification if the set of its achievable executions satisfies the predicate for that component.

Again, the predicate is defined for a set of executions, for many of the same reasons as before. In addition, if the specification is a ''loose'' one, admitting many possible implementations, there may be the additional requirement that we be able to identify sets of executions that belong to a single implementation.

A specification may be as simple as a set of executions defining a component or system, in which case the methods of specification are no different from those used for executions. In this case, there is effectively only a single possible realisation of the specification, and the specification itself is exactly the set of executions of the component. Usually, though, such a specification would be considered too restrictive.

It is more common that the executions permitted by a specification allow for several alternative implementations. In fact, it may be impossible to construct a component which can engage in all the executions permitted by a specification. A specification, then, may either state or imply a collection of satisfactory execution sets corresponding to the collection of permitted implementations.

If such is the case, it may then be important to know whether two executions permitted by a particular specification can both be produced by a single implementation of the component. This may be necessary if the system security property we are trying to demonstrate is one that considers relationships between system executions rather than one execution at a time. If the property allows execution X only if execution Y is also possible, it would be invalid to consider this satisfied if X and Y were produced by two different implementations.

On the other hand, the satisfaction of some specifications can be determined by inspecting the executions one at a time. Often, it is convenient to think of this kind of specification as a combination of *safety* properties and *liveness* properties. A safety property is a prohibition against certain actions, while a liveness property is one that requires particular actions. A collection of executions (i.e., an implementation) satisfies a safety property if every member execution satisfies it. A collection satisfies a liveness property if at least one member satisfies it.

With regard to the properties themselves, [Abadi90] distinguishes them formally in this way: a property P is a safety property iff, for any behaviour b that satisfies it, all finite prefixes of b also satisfy it. This fits our intuitive definition, since b being void of any objectionable acts implies that parts of it, prefixes in particular, must be similarly clean. Another property Q is considered a liveness property iff, for all finite behaviour prefixes p, there exists at least one conceivable extension of p that satisfies Q. This corresponds to a requirement that ''eventually'' some desired thing happens, and that it is ''never too late''. Some notions of liveness are more demanding than this, imposing deadlines for compliance rather than waiting indefinitely.

It is possible that a specification may allow (but not require) some executions that are not implementable. If no implementation can exhibit them, these behaviours cannot threaten security, but their inclusion in the specification may permit it to be expressed more simply. At first glance, this seems to conflict with McCullough's observation [McCullough88c] that

> If one makes a system more deterministic [they] may invalidate the security proof, since it may produce new correlations between high-level inputs and low-level events. This problem *is inherent in any definition of security for nondeterministic systems.*

(italics mine), but the models he considers do not distinguish between permitted and required behaviours. The situation McCullough describes is one where the implementer's view of the specification is that it permits but does not require, while the security definition used implicitly requires all the specified behaviours. If we want our security models to be useful with specifications, we must avoid this kind of confusion. The models must distinguish between ''required'' and ''permitted'' behaviours, and this distinction must be accounted for in the definitions of the properties we wish to enforce. So, for example, if a desired property of an implementation is that *it may behave like X only if behaviour Y is also displayed*, this should be modified when we apply it to specifications, to read *behaviour X is permitted only if behaviour Y is required in the same implementation*.

Specifications may only partially specify executions, such that there exist properties of any implementation that are not discussed. Confidentiality properties are a good example of this since they are strictly safety properties and do not imply any liveness. This is ultimately true even of the ''Xand Y''type of properties we have been discussing, since if there is no ''X''then there is no requirement for a ''Y'',and so they would be satisfied by a system that does nothing at all.

The existence of such a partial specification does not imply that other concerns are of no interest, simply that they may be determined by some other partial specification, so that an ultimately acceptable implementation is one that simultaneously satisfies several sub-specifications. The likely benefit here is that different constraints may be more conveniently or more effectively presented using different modelling techniques. Also, combining two or more types of constraints in a single model may result in their being linked together in ways that reduce overall flexibility.

The total requirements for a device may include a specification of its functionality, a set of standards concerning the methods and procedures used in its design and construction, perhaps some physical restrictions on size, weight, and other environmental factors, as well as cost limits. The kinds of specifications that are the focus of this report are functional ones, and should be cleanly separated from any non-behavioral restrictions. This implies that any fundamental definition of secure behaviour, such as confidentiality or integrity, should be based entirely on the external behavioural properties of a component, and that concerns about the internals or method of construction of the component be addressed by a companion specification. It is for this reason that we require a representation that does not depend on the internal structure or operation of the specified component.

8.4.1.4. Decompositions

These functions reflect the composition of the system by relating the system and component executions. Each collection of N submappings represents a single legal interconnection of the N components. Given a set of candidate system executions, the mappings take them to the corresponding sets of component executions, which are then required (in the definition of composable security) to satisfy the specifications of the various components. The effect of the quantifications over D and for all satisfactory E is to require that, for every permitted interconnection, each set of achievable system behaviours is secure. There may be more than one such set for a given interconnection, because one or more components may be only loosely specified, and each implementation of each component may give rise to a different set of system behaviours.

This decomposition-based approach does imply that the system execution contains enough system-internal information to allow the reconstruction of the component executions, in at least sufficient detail that it may be determined whether or not the proposed component specifications permit the candidate behaviour. It is likely that not all of this detail is of interest to the system security objective S, so the first step in applying that predicate would probably involve some sort of projection operation that hides all of this detail, but it is not necessary to expose that step at this stage.

One concern is that this formulation requires the prior enumeration of all legal (de)compositions, when what was really wanted here was an assurance that following some simple rules for composition would guarantee a secure composite system. The difficulty is that, unless we are willing to say something definite at this point about what kinds of ‘‘ports’’ and ‘‘signals’’ and ‘‘connections’’ we are talking about, and what kind of ‘‘simple rules’’ we envision, we really have no good way of introducing those notions here. We are saved here by the fact that there are a finite number of components connectable in (we have decreed) a finite number of ways, and so the enumeration is possible, at least in principle. It is expected that particular applications of this modelling style, such as a model of confidentiality-security, will be able to make those more definite statements and so to encompass the ‘‘simple rules’’ directly.

8.4.2. Application to Confidentiality

To make this definition correspond to our earlier diagram (figure 8.6), we can restrict S to be a confidentiality property, and consider four specifications:

t_0 is a specification that includes restrictions on the ML system of interest.

t_1 is identical to t_0, and describes the "other ML" component.

t_2 and t_3 are, respectively, the High and Low component specifications. They are equal and permissive, i.e. they make no specific demands on the component executions.

The decompositions D must be restricted to those in which the High component communicates with the ML ones, the ML talk to each other, Low communicates with ML, and the Low component may write up to the High one.

The only obvious flaw here was brought up in the original informal discussion of the closed model. It is that we are implicitly relying on the composability of the ML specification when we specify a single "other ML" system. This occurs because our original objective was to demonstrate closed system security for a system composed with an arbitrary number of High, Low, and Other ML components, but the definition above only allows for a fixed number of components. We rely on the aggregation properties of the multiple High and Low components to collect them into one ‘‘super-component’’ of each type. We also assume that a collection of ML systems may be represented by a single aggregate. In the general case, this assumption would have to be separately verified. However, in the specific case of confidentiality, it may be possible to show that a large class of composably S-secure ML specifications are necessarily piecewise composable (i.e., can be aggregated) with other ML components.

9 A SPECIFICATION APPROACH

Thus the theory of description matters most.
Wallace Stevens,
Description Without Place,
1947.

It is written in the language of mathematics,
and its characters are triangles, circles, and other geometrical figures,
without which it is humanly impossible to understand it;
without these, one is wandering about in a dark labyrinth.
Galileo Galilei,
Il Saggiatore,
1623.

9.1. INTRODUCTION

In order to speak precisely of the security attributes of a system, we must develop precise notions of what ''security'' and a ''system'' really are. This section presents a system model that we believe to be appropriate for examining questions of security, and for dealing with issues of composition. This model may be applied to systems at any level of abstraction, which means that we can examine security issues of widely varying form in the context of systems defined at very abstract or very concrete levels.

Having a model and security criteria that are ''portable'' across a wide range of abstraction ensures that our results can be applied to high-level security objectives, circuit-level building blocks, and all descriptions in between. Therefore we make no real distinction here between the notions of *specification* and *implementation*, except that we recognize that the former are generally less detailed than the latter. For example, a subroutine coded in a programming language, usually considered an implementation to fit some more abstract purpose, can be seen as a specification that a compiler must adhere to when generating code in a lower-level language.

9.2. SYSTEM MODEL

We begin by describing the formalisms used to model systems and their characteristics. The following definitions are largely from [Abadi88] and

[Abadi90b]; however, some of our definitions and notations differ, since neither of these reports addresses security concerns.

The systems we are modeling here are nondeterministic state machines. They are described by means of *behaviours*, which are individual system histories, each representing one possible execution of a system. These behaviours consist of an *initial state*, representing the state of the system at its creation, and an infinite number of *behaviour steps*, which describe the transitions of the system from state to state. Each step not only describes the state reached, but also the *agent* responsible for causing the step. Identifying the agent at each step allows the development of models representing the concurrent operation of multiple systems, each agent representing the actions of an individual system. A behaviour thus records the sequence of states traversed by a system execution, as well as the agent responsible for each transition.

Inputs and outputs are encoded in the state at each step; if a particular state variable changes value over a step, this can be regarded as an output by that step's agent, while other agents may consider the state variable change to be an input. A step need not result in the change of system state, however. Such a step is referred to as a *stuttering step*, in which case we may leave the agent responsible for this step undefined. We say that two behaviours are *stuttering-equivalent* if removing all stuttering steps from each leaves identical behaviours.

A system is modeled by the set of behaviours that describe each possible complete execution of the system. We make the additional requirement that this set is closed under stuttering, which means that, for each behaviour in the set, all behaviours stuttering-equivalent to that behaviour are also in the set. A stuttering-closed set is referred to as a *property*. Since an infinite number of behaviours are stuttering-equivalent to any behaviour, properties contain an infinite number of behaviours.

We are interested here in the design of components of concurrent systems. This rules out the establishment of any finite state set in which our machines are to operate, since the addition of a new system to the composite can introduce arbitrary new state variables. Therefore we assume an infinite state space, though each property is defined in terms of some subset of this space. Similarly, we assume an infinite agent space, since we cannot put an upper bound on the number of components in a concurrent system. Each property then describes the actions of a subset of the infinite agent space.

This formalism models systems by clocked state machines, and simulates concurrency by the interleaving of steps from different systems. To allow flexibility in implementing specifications in this formalism, we generally allow at most one state variable to change value per behaviour step. We assume this in the discussion below, and justify it by arguing that changing multiple state variables per step is a form of true concurrency, which we are replacing by interleaving.

9.2.1. Notation

Behaviours will be represented by lower-case Greek letters. Given some behaviour σ, we denote by σ_i the i^{th} behaviour step. The notation $\sigma_{i \to j}$ denotes a *behaviour segment*, the subsequence of σ from the i^{th} through j^{th} steps, inclusive. Note that $\sigma_{i \to j}$ is not a behaviour unless j is infinite; removing the first i steps from a behaviour σ leaves the behaviour $\sigma_{i \to \infty}$. The notation σ can then be seen as an abbreviation for $\sigma_{0 \to \infty}$.

The agent portion of a behaviour step may be extracted by $\mathbf{a}(\sigma_i)$. This operator can also be applied to behaviours and behaviour segments to yield a sequence of agents: for example, the sequence of agents responsible for behaviour σ is represented by $\mathbf{a}(\sigma)$. If σ_i is a stuttering step, we may write $\mathbf{a}(\sigma_i) = \perp$. We refer to the agent set which behaves according to property P as μ^P; the remainder of the infinite agent set, whose actions are not restricted by P, is denoted $\neg\mu^P$.

Similarly, the state portion of a behaviour step may be extracted by $\mathbf{s}(\sigma_i)$. The notation $\mathbf{s}(\sigma_{i \to j})$ represents the sequence of states in the behaviour segment $\sigma_{i \to j}$, and the initial system state is $\mathbf{s}(\sigma_0)$.

The notation $\underline{\sigma}$ represents the behaviour σ with all stuttering steps removed. The notation $\sigma_{i \hat{\to} j}$ represents the behaviour formed by appending an infinite number of stuttering steps to the tail of behaviour segment $\sigma_{i \to j}$. Thus $\underline{\sigma}$ may or may not be infinite in length, depending on whether or not it contains an infinite number of non-stuttering steps. On the other hand, $\sigma_{i \hat{\to} j}$ is always infinite in length.

The behaviour $\sigma_{0 \hat{\to} j}$ will be referred to as a *stutter-prefix* ; a behaviour prefix converted into a behaviour by appending an infinite stuttering suffix. Intuitively, a behaviour with an infinite number of stuttering steps at its tail represents a machine that has halted.

9.2.2. The Logical Form of Properties

We have attached a precise meaning to the words *behaviour* and *property* for the purposes of this investigation. This is the same terminology as in [Abadi90b]. It is important to remember that properties are only sets of behaviours of nondeterministic state machines. These sets are defined by predicates on the universe set of behaviours, U.

In practice, temporal logic is used in the definition of properties. This allows behaviours to be included or excluded from properties based on relations among values of state components, and the identities of agents, at different points in the system lifetime. Writing concise, correct formulas for even simple properties is surprisingly subtle and tricky. We will describe here a notation that is derived from Lamport's ‘‘Temporal Logic of Actions’’

[Lamport83] [Lamport89] [Lamport90], but modified for reasoning about composition, and to suit our definitions above. Much of the formalism to follow has appeared in [Abadi90a]; we adopt a less rigourous approach and make a few alterations.

We have described properties as sets of behaviours. When a property is expressed as a logical formula, this set-membership relation corresponds to the *satisfaction* or *modeling* of the formula by a behaviour. We can define the *satisfaction relation* between behaviours and arbitrary properties inductively via a set of rules describing the meaning of formulas in their various forms. More detail on this style of semantic description can be found in [Abadi90a].

The simplest formula is the *state predicate*, which consists of relationships among state variables that must hold at some step in the behaviour. A behaviour satisfies (models) a state predicate property if the state predicate holds in the initial state of the behaviour. This is written:

$$\sigma \models P \triangleq \mathbf{s}(\sigma_0) \models P,$$

where P is a state predicate, and the symbol $\models$ can be read as ''models'', or ''satisfies''. Thus a property whose formula consists only of a state predicate contains all behaviours whose initial states satisfy that predicate. For example, the property defined by:

$$x=0 \wedge y=1$$

consists of all behaviours with state variable x equal to 0 and state variable y equal to 1 initially.

Another basic formula makes use of a modified type of state predicate known as an *action*. An action consists of relationships among *primed* and *unprimed* state variables, where unprimed variables designate variables in one state, and primed variables designate variables in the next. Thus, actions express relationships that must hold among state variables across some step. A behaviour satisfies an action if its first step satisfies the action:

$$\sigma \models A \triangleq (\mathbf{s}(\sigma_0), \mathbf{s}(\sigma_1)) \models A,$$

where A is an action, the values for unprimed variables in A come from $\mathbf{s}(\sigma_0)$, and the values for primed variables in A come from $\mathbf{s}(\sigma_1)$.

Since an action is the work of some agent, we include the agent's identity by using the [ ] operator:

$$\sigma \models [A]_a \triangleq \mathbf{a}(\sigma_1)=a \Rightarrow \sigma \models A.$$

For example, the property defined by

$$[x'=x+1]_g$$

consists of all behaviours in which state variable x is incremented in the first step by agent g.

We say that a behaviour satisfies the state-predicate *Enabled* (A), for any action A, if the values of the unprimed variables in A hold in the initial state.

For example, *Enabled* $(x=0 \wedge y'=y+1)$ holds for all behaviours with $x=0$ in the initial state.

More general temporal formulas are constructed using the ''henceforth'' operator, defined by:

$$\sigma \models \Box F \triangleq \forall i \geq 0 : \sigma_{i \to \infty} \models F,$$

where F is an arbitrary formula, of the kind we are now describing. Intuitively, $\Box F$ means that the formula F must hold for all behaviour suffixes. For example, the property defined by

$$\Box (x=y)$$

consists of all behaviours in which state variables x and y have the same value in every state, and

$$\Box [z'=x-y]_r$$

consists of all behaviours in which every step by agent r sets state variable z equal to the difference of x and y.

Temporal formulas can be logically composed in the obvious ways. The necessary definitions are complement:

$$\sigma \models \neg F \triangleq \neg(\sigma \models F),$$

and conjunction:

$$\sigma \models F \wedge G \triangleq \sigma \models F \text{ and } \sigma \models G.$$

For example, the property defined by

$$\neg (x=0 \wedge \Box [x'=y]_q)$$

consists of all behaviours in which state variable x is initially not 0, or some step by agent q does not set x equal to the previous value of y.

These basic formula types (predicates and actions) and operators ([ ], $\Box$, $\neg$, and $\wedge$) constitute a powerful specification language. It is useful to define some derived operators as well. Two simple logical operators are disjunction:

$$\sigma \models F \vee G \triangleq \neg(\sigma \models \neg F \text{ and } \sigma \models \neg G),$$

and implication:

$$\sigma \models F \Rightarrow G \triangleq \sigma \models \neg F \vee \sigma \models G.$$

Another useful operator is $< >$:

$$\sigma \models \langle A \rangle_a \triangleq \neg \sigma \models [\neg A]_a,$$

which expands to:

$$\sigma \models \langle A \rangle_a \triangleq \mathbf{a}(\sigma_1)=a \wedge \sigma \models A.$$

Thus a behaviour models $\langle A \rangle_a$ if its first step is taken by agent a, and the step models A.

A useful derived temporal operator is ‘‘eventually’’, $\blacklozenge$:

$$\blacklozenge F \triangleq \neg \Box \neg F.$$

By applying the identity $\forall P \equiv \neg \exists \neg P$, $\blacklozenge$ can also be defined as:

$$\sigma \models \blacklozenge F \triangleq \exists i \geq 0 : \sigma_{i \to \infty} \models F,$$

which means that F must hold for *some* behaviour suffix.

The temporal operator ‘‘leads to’’, $\to$, can be defined by:

$$F \to G \triangleq \Box(F \Rightarrow \blacklozenge G).$$

In words, $F \to G$ means that, for all time, the occurrence of F is eventually followed by G.

We have defined properties in two ways: first as sets of behaviours, and second as formulas in a temporal logic. Thus we can use the notations $\sigma \models P$ and $\sigma \in P$ interchangeably. When discussing the combination of multiple properties, without reference to their individual form, set notation is more convenient. Formulas resulting from logical operations on predicates have the following analogs in terms of set operations on properties:

$$\sigma \models \neg P \equiv \sigma \in (U - P),$$

(logical negation corresponds to set complement),

$$\sigma \models P \wedge Q \equiv \sigma \in (P \cap Q),$$

(logical conjunction corresponds to set intersection),

$$\sigma \models P \vee Q \equiv \sigma \in (P \cup Q),$$

(logical disjunction corresponds to set union), and

$$\sigma \models P \Rightarrow Q \equiv \sigma \in ((U - P) \cup Q),$$

(logical implication corresponds to behaviours that either fail to satisfy the premise or satisfy the conclusion).

9.2.3. Safety and Liveness Properties

Properties can themselves be classified in many ways. Two important kinds are *safety* and *liveness* properties [Lamport83] [Alpern85]. In the next section we will introduce some different kinds of properties specialized for security applications.

A property P is a safety property if and only if it satisfies the following formula:

$$\forall \sigma \in P \ \ \forall k \in \mathbb{N} \ \ \exists \gamma \in P : (\hat{\sigma}_{0 \to k} = \gamma),$$

where $\mathbb{N}$ is the set of natural numbers. That is, a safety property P contains every stutter-prefix of every behaviour in P. If P is a safety property, we write

$$P \in \mathsf{S},$$

where S is the set of all safety properties.

A property P is a liveness property if and only if it satisfies the following formula:

$$\forall \sigma \in U \ \ \forall k \in \mathsf{N} \ \ \exists \gamma \in P : (\sigma\hat{_{0\to k}} = \gamma\hat{_{0\to k}}).$$

That is, every stutter-prefix of every behaviour in U, is a stutter-prefix of a behaviour in P. If P is a liveness property, we write

$$P \in \mathsf{L},$$

where L is the set of all liveness properties.

The intuition behind safety properties is that they represent requirements on a system that must hold at all times during the system's operation. Therefore, every stutter-prefix of every *safe* behaviour must itself be safe. Liveness properties, on the other hand, are requirements that a system eventually does some specific thing, regardless of what it has done previously. Therefore, every finite initial portion of a system behaviour must be extendable to a *live* behaviour.

The definition above asserts that a safety property is a closed set under the relation of *prefix-equivalence*. Two behaviours are prefix-equivalent if one is a stutter-prefix of the other, and we say that a safety property is a *prefix-closed* property. Liveness properties are not closed sets, but using these definitions it can be shown [Alpern85] that every property can be expressed as the intersection of a safety and a liveness property. That is,

$$\forall P \in \mathsf{P} \ \ \exists S \in \mathsf{S} \ \ \exists L \in \mathsf{L} : (P = S \cap L),$$

where P is the set of all properties.

It can also be shown that safety properties (i.e. the set S) are closed under the operations of union and intersection, and liveness properties (the set L) are closed under union. This means that the union or intersection of two safety properties is a safety property, and the union of two liveness properties is a liveness property.

9.2.4. Specifications

A *specification* is a special kind of property, defined in [Abadi90b] as having the form

$$S = (E \Rightarrow M)$$

where E is the *environment* property, and M is the *system* property. A specification is the set of behaviours that represent correct system operation, *assuming* reasonable environment behaviour. Thus a specification includes all behaviours that either satisfy the system property, or fail to satisfy the environment property. The only behaviours disallowed by the specification are thus behaviours that both satisfy the environment property and fail to satisfy the system property; these represent histories in which the environment performed as anticipated, but the system responded incorrectly.

A specification describes the actions of the entire infinite agent set, by partitioning it into two infinite subsets: system agents, whose allowed behaviours make up the system property, and environment agents, whose allowed behaviours make up the environment property. Each behaviour step in a specification can then be classified as a *system step*, or an *environment step*. Specifications are thus descriptions of systems that can be used as components in an arbitrary environment of other components.

Given two specifications S and S', S' can be said to *implement* S if

$$S' \subseteq S,$$

where $\subseteq$ is the ''subset-of'' relation [Abadi88]. This means that every behaviour allowed by S' must be allowed by S. Note that S' may disallow some behaviours allowed by S, but this is not a problem as long as S is a specification as defined above.

The implication of this definition of *implement* is that the scope of the term *specification* is more restricted than we may be accustomed to. For example, a specification S cannot include the requirement that any implementation S' be a safety property, even if S is itself a safety property, since S' is allowed to implement any subset of S. The logical specification S must be expressed by the conjunction and disjunction of logical formulas in the state variables of the system of interest. Thus, if S' is truly required to be a safety property, this must be demonstrated separately from any proof that $S' \subseteq S$.

In practice, one rarely requires that $S' \in \mathsf{S}$, since real specifications generally involve some liveness requirement, and allowing all stutter-prefixes violates any non-trivial liveness requirement. Even less likely is that $S' \in \mathsf{L}$, since this allows arbitrary system behaviour for an arbitrary (but finite) number of steps. However, we will see later that security considerations may involve requirements that are similar in form to safety and liveness. These are expressed as constraints on the form of S' as a whole, and thus cannot be included as a conjunct of S.

The implementation S' is generally more detailed than S, requiring many internal state variables to accomplish the goals specified more abstractly in S. For the behaviours of S and S' to be compatible, S' must be written using existential quantification over its internal variables. Also, the more detailed machine in S' may take many steps, involving change to only the internal variables, for any single step in S. Thus S' will appear to stutter when its externally-visible variables alone are considered. For S and S' to be comparable, then, they must be stuttering-closed. Moreover, since each behaviour step can involve at most one agent, any *concurrency* that appears to occur must actually be an *interleaving* of steps from different subsystems. Thus the behaviours of each subsystem must be capable of pausing to allow a behaviour step from another subsystem. Since an arbitrary number of subsystems may be connected together, properties must be closed under arbitrary amounts of stuttering.

Given n compatible specifications, what is the specification of their composition? It is clear that any behaviour of the composition must also be a behaviour of each component, implying that the behaviour belongs to the intersection of the component specifications. That is,

$$S = S_1 \cap S_2 \cap \cdots \cap S_n,$$

where S is the specification of the composite system, and S_i is the specification of the i^{th} component.

The main result of [Abadi90b] is a method for proving that the composition of multiple implementation modules can satisfy a single higher-level specification. That is, sufficient conditions are provided to ensure that, given components $S_1, S_2, \ldots, S_n$, and specification S,

$$S_1 \cap S_2 \cap \cdots \cap S_n \subseteq S.$$

The i^{th} component in a composite system has the specification $S_i = (E_i \Rightarrow M_i)$, leading to the following requirement for the composition of subsystems to satisfy a high-level specification:

$$(E_1 \Rightarrow M_1) \cap (E_2 \Rightarrow M_2) \cap \cdots \cap (E_n \Rightarrow M_n) \subseteq (E \Rightarrow M).$$

Here, E is the environment property of the specification, representing the operating environment of the composite system as a whole. Each E_i, on the other hand, is an environment property for a component in the context of all the remaining components. Each component's environment consists of a portion of the overall system's environment, as well as the system properties of all directly connected components.

For the composite system to work correctly, we must require that each component's environment property is satisfied. We therefore have the following requirement, for the i^{th} component:

$$E \cap M_1 \cap \cdots \cap M_n \subseteq E_i.$$

This means that all behaviours that satisfy the high-level environment, as well as all of the components' system properties, must satisfy each individual component's environment property as well. Since we want this to hold for all components, we can combine all n of these requirements into one:

$$E \cap M_1 \cap \cdots \cap M_n \subseteq E_1 \cap \cdots \cap E_n. \tag{1}$$

Fundamental to the design (or analysis) of systems by composition is the assumption that the functionalities of the components wholly determine the functionality of the system at a higher level of abstraction. In other words, the high-level system property is exactly the composition of the component system properties, or:

$$M = M_1 \cap \cdots \cap M_n.$$

With this simplification, the high-level specification S becomes the requirement that behaviours satisfying the high-level environment property E also satisfy the composition of the components system properties M_i, and the composition requirement is:

$$(E_1 \Rightarrow M_1) \cap (E_2 \Rightarrow M_2) \cap \cdots \cap (E_n \Rightarrow M_n) \subseteq (E \Rightarrow M_1 \cap \cdots \cap M_n). \quad (2)$$

If the real S of interest is not of the form $E \Rightarrow M_1 \cap \cdots \cap M_n$, then additional work will be needed after proving the composition requirement. However, it appears that (2) succinctly captures the essence of the correctness of composed systems. The *Composition Principle* of [Abadi90b] is simply ‘‘(1) implies (2)’’, but in fact this unqualified statement is not valid in general. A few, quite natural, conditions must hold on the environment and system properties, but we will not give the details here. Instead, we will begin considering security issues.

9.3. SECURITY PROPERTIES

It is our contention that the formalism just described provides a convenient language for expressing some of our concerns with regards to the composition of secure systems. If the ‘‘security’’ of a system could be expressed as the logical combination of safety and liveness properties, the results of [Abadi90b] would directly apply, giving us a simple (in principle) rule for deducing the security of a composite system from the security of its components. However, this is not the case.

Below, four popular definitions of confidentiality are translated into our formalism. These provide a good demonstration of the expressiveness of this approach. We assume, for simplicity, that a system’s externally visible state variables can be partitioned into two subsets: high-level, denoted by **H**, and low-level, **L**. Generalization of this labeling scheme to multiple levels of confidentiality entails, in most cases, a straightforward extension of the concepts presented. In some cases we further partition **H** and **L** into low-input, low-output, high-input, and high-output variable sets, and name these **LI**, **LO**, **HI**, and **HO**, respectively. For example, the notation **LI** $(\sigma_{i \rightarrow j})$ represents the sequence of low-input states in the behaviour segment $\sigma_{i \rightarrow j}$.

9.3.1. Non-Deducibility

The first confidentiality property we will discuss is Non-Deducibility [Sutherland86]. As with our definitions of safety and liveness, we say that a property P is a *non-deducible* property if and only if it satisfies the following formula:

$$\forall \alpha \in P \quad \forall \beta \in P \quad \exists \gamma \in P \quad \forall t \geq 0 :$$

$$(\mathbf{a}(\alpha_t) = \bot \vee \mathbf{a}(\beta_t) = \bot) \Rightarrow (\mathbf{L}(\alpha_t) = \mathbf{L}(\gamma_t) \wedge \mathbf{HI}(\beta_t) = \mathbf{HI}(\gamma_t)) .$$

This definition, minus the left side of the implication, is similar in form to that given in [Thomson88] and [Millen90]. If P is a non-deducible property, we write

$$P \in \mathsf{ND},$$

where ND is the set of all non-deducible properties.

Note that non-deducibility, like safety, is a *closure* requirement — given any two behaviours α and β in P, a behaviour γ must be in P which shares the low input-output activity of α and the high input activity of β, except following steps that are non-stuttering in both α and β. This ensures that any observations made by a low-level user do not rule out any high-level input sequence. We may also say, as with our definitions of prefix-closure and stuttering-closure, that an ND property is ND-closed.

We make no requirement on γ's state following steps that are non-stuttering in both α and β, because input and output changes are the result of steps by distinct agents, and each step of a behaviour may involve at most one agent. Without this concession, it is necessary, in general, to show the existence of a behaviour γ that changes its low output state and high input state on the same step. Non-deducibility would then be unsatisfiable for any system with high inputs and low outputs!

Now we must consider whether this pragmatic use of the implication in our definition of non-deducibility has weakened it with respect to previous formulations. It has not, since we are dealing with properties, which must be stuttering-closed, and the occurrence of stuttering is not observable to the user. Thus, for every pair of behaviours α and β which take non-stuttering steps simultaneously, there are an infinity of behaviours which are identical from an observer's point of view, but which *do not* take non-stuttering steps simultaneously. A behaviour γ must exist for these behaviours which has the appropriate state on all steps. The low-level user is thus prevented from making any deduction of the high-level input sequence.

It is worth noting that we had originally attempted to avoid these formal difficulties by weakening the equivalence requirements in the property formulation. An earlier version of non-deducibility was:

$$\forall \alpha \in P \quad \forall \beta \in P \quad \exists \gamma \in P : \underline{\mathbf{L}}(\alpha) = \underline{\mathbf{L}}(\gamma) \wedge \underline{\mathbf{HI}}(\beta) = \underline{\mathbf{HI}}(\gamma) .$$

This formula involves the comparison of stutter-free sequences, which effectively allows γ to take two steps whenever α and β take non-stuttering steps simultaneously. Now, to prove non-deducibility, we are left with the task of showing the existence of a behaviour γ that changes it low output and high input states on successive steps, which is at least achievable in principle.

The problem with this definition is that stuttering-equivalence materially alters the meaning, because there is no longer any requirement that γ respect the *interleaving* of steps from α and β. That is, the system is no longer required to support the *simultaneous* occurrence of arbitrary low input-output and high input sequences. Earlier definitions of non-deducibility (for example, [Goguen84] and [Sutherland86]) have similarly disregarded the importance of interleaving, but it is an important consideration. See [Guttman88], and section 5.4.2 of this report, for a discussion of the necessity of protecting the interleaving of high and low events.

Of lesser importance is the fact that *proving* the stuttering-equivalence of two behaviours (or state sequences derived from these behaviours) is substantially harder than proving full equivalence. This is because two sequences are equivalent if and only if each of their equal-length prefixes are equivalent, so induction on sequence lengths can be used to prove equivalence of infinite sequences. However, the stuttering-equivalence of two sequences does not imply that any equal-length prefixes are stuttering-equivalent, rendering inapplicable the proof by induction technique.

9.3.2. Non-Deducibility on Strategies

Following [Wittbold90], we define a *strategy* as a non-deterministic (not single-valued) function $\mathbf{f}$ from a user's input/output history to that user's next input. Here, we are concerned with the strategies of high-level users, and define an $\mathbf{f}$-*trace* as a possible behaviour resulting from the application of high-level strategy $\mathbf{f}$, that is, behaviour σ is an $\mathbf{f}$-trace if and only if

$$\forall\, t > 0 : \mathbf{f}\,(\mathbf{H}(\sigma_{0 \to t-1})) = \mathbf{HI}(\sigma_t)\ .$$

We now define F^S as the set of all $\mathbf{f}$-traces for all strategies $\mathbf{f}$ in system S. Thus F^S characterizes all possible high-level interactions with S, embedded in S-behaviours that are not confined to P. Clearly, this is a large set, describable only in a very abstract sense.

We say that a property P is a *non-deducible-on-strategies* property if and only if it satisfies the following formula:

$$\forall\, \alpha \in P \quad \forall\, \beta \in F^S \quad \exists\, \gamma \in P \quad \forall\, t \geq 0 :$$

$$(\mathbf{a}(\alpha_t) = \perp \vee \mathbf{a}(\beta_t) = \perp) \Rightarrow (\mathbf{L}(\alpha_t) = \mathbf{L}(\gamma_t) \wedge \mathbf{HI}(\beta_t) = \mathbf{HI}(\gamma_t))\ .$$

If P is a non-deducible-on-strategies property, we write

$$P \in \mathsf{NDS}.$$

Again, NDS is a closure requirement — given any behaviour α in P, and any $\mathbf{f}$-trace β, a behaviour γ must be in P which shares the low input-output activity of α and the high input activity of β. Note that the definitions of ND and NDS are nearly identical. The latter effectively replaces universal quantification over the ''high input space'' of P with universal quantification over the ''high-level strategy space'' of S. This strategy space is much larger, of course, because it is not restricted by the definition of P (in effect, it represents the computation of any feasible S-machine). This generality complicates the application of NDS in practice, but accounts for the composability of NDS properties.

9.3.3. Non-Interference

The Non-Interference model of confidentiality introduced in [Goguen84] was generalized in [McCullough87] to apply to non-deterministic systems. We present a definition of non-interference here, based on that in [McCullough87], to illustrate the various forms that security properties can take. The following logical formula is not, however, the only possible transcription of the non-interference property into our model.

We say that a property P is a *non-interference* property if and only if it satisfies the following formula:

$$\forall \alpha \in P \quad \forall \beta \in U^S \quad \forall t > 0 \quad \exists \gamma \in P :$$

$$\mathbf{L}(\alpha) = \mathbf{L}(\beta) \wedge \mathbf{HO}(\alpha) = \mathbf{HO}(\beta) \wedge \mathbf{HI}(\alpha_{0 \to t-1}) = \mathbf{HI}(\beta_{0 \to t-1})$$

$$\Rightarrow \mathbf{L}(\beta) = \mathbf{L}(\gamma) \wedge \mathbf{HI}(\beta) = \mathbf{HI}(\gamma) \wedge \mathbf{HO}(\beta_{0 \to t-1}) = \mathbf{HO}(\gamma_{0 \to t-1}) .$$

This is another closure requirement, complicated by the requirement to enumerate all possible *perturbations* of a behaviour in P (effected here by choosing β from all behaviours in the S-universe, then eliminating those which are not perturbations in the left-side of the implication). If P is a non-interference property, we write

$$P \in \mathsf{NI}.$$

9.3.4. Flow Model

The Flow Model, introduced in [McLean90] and more fully developed as part of a probabilistic model of security in [Gray91], is interesting because of its treatment of statistical information flow, as well as for its composability [Gray91].

To begin, we must define the notion of the probability of a behaviour set, $\mathbf{p}(B)$. This is the probability of some behaviour in B occurring, given that the probability of occurrence of some behaviour in the universe is unity, that is, $\mathbf{p}(U^S)=1$. Note that, since the universe property, like all properties, contains an infinite number of behaviours, the probability of occurrence of any single behaviour is infinitesimal. Also note that our probabilities are on behaviours sets, not just properties (and thus need not be stuttering-closed).

We say that a property P is a *flow-model* property, $P \in \mathsf{FM}$, if and only if it satisfies the following formula:

$$\forall t > 0 \quad \forall \alpha \in P \quad \forall \beta \in P \quad \forall \gamma \in P \ :$$

$$\mathbf{p}(L_s \cap H_s) > 0 \ \Rightarrow \ \mathbf{p}(L_t \mid L_s \cap H_s) = \mathbf{p}(L_t \mid L_s) \ ,$$

where

$$L_t = \{\sigma \in P \ : \mathbf{L}(\sigma_t) = \mathbf{L}(\alpha_t)\} \ ,$$

$$L_s = \{\tau \in P \ : \mathbf{L}(\tau_{0 \to t-1}) = \mathbf{L}(\beta_{0 \to t-1})\} \ ,$$

$$H_s = \{\upsilon \in P \ : \mathbf{H}(\upsilon_{0 \to t-1}) = \mathbf{H}(\gamma_{0 \to t-1})\} \ .$$

This formula asserts that the probability of occurrence of any low-level state is independent of the preceding high-level state sequence. The notation $\mathbf{p}(A \mid B)$ is read as "the probability of A given B", and is computed as:

$$\mathbf{p}(A \mid B) = \frac{\mathbf{p}(A \cap B)}{\mathbf{p}(B)} .$$

The requirement that the denominators of these conditional probabilities be non-zero is the reason for the $\mathbf{p}(L_s \cap H_s) > 0$ requirement above.

9.3.5. Composing Confidentiality

The diagram in figure 9.1 shows the topology and rough functionality of the composite system formed by two systems T and D. T outputs as signal z the bit-wise exclusive-or of its two inputs i and k, while D generates a random bit-stream k, and outputs as l the bit-wise exclusive-or of z and k. From a security perspective, D is functioning as an encryption device, and T a Trojan horse, as will be explained below.

For simplicity we consider only two sensitivity levels on data; for this system signals i, k, and z are high-level, and all others are low. The signals labeled c_i, etc., are handshaking clock lines for synchronizing the operation of the composite system. Any clocking scheme which ensures that a system only reads an input once it has been written, and does not generate an output until the previous value has been read, could be used here. This particular scheme will be fully elaborated below, but we will see that, aside from ensuring that the system operates in a reasonable manner, it is of no consequence

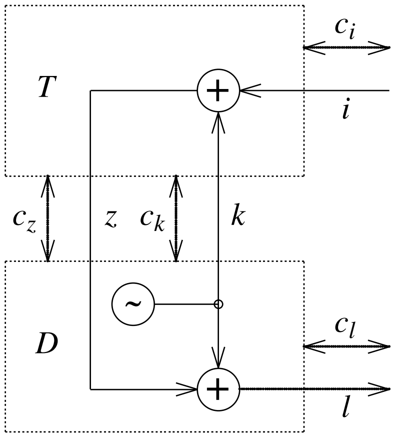


Figure 9.1: A Composite System.

from a security standpoint.

Since we are interested in the precise functionality of this system, we will describe it below using *programs* , which unambiguously specify the behaviour of T and D, at least to the level of detail necessary for investigation of security functionality. In fact these programs are more deterministic than necessary (which is to say, they constrain the implementations more than one might like), but they are simple and correct. We can view these programs as an implementation of the system informally described by figure 9.1 and the text of the previous paragraphs.

Nevertheless, these programs are true specifications, which can be written in the logical clause form used in this chapter and elsewhere. The program form is semantically equivalent, but much more intuitive for the human reader accustomed to reading computer programs.

The specification for system T is

$$T = (E^T \Rightarrow M^T) ,$$

where E^T is described by:

$$
\begin{array}{l}
c_z = 0 \\
c_i = 0 \\
c_k = 0 \\
\text{loop} \\
\quad \text{if } c_i = 0 \\
\quad\quad i = ? \\
\quad\quad c_i = 1 \\
\quad \text{if } c_k = 0 \\
\quad\quad k = ? \\
\quad\quad c_k = 1 \\
\quad \text{if } c_z = 1 \\
\quad\quad c_z = 0
\end{array}
$$

and M^T is described by:

$$
\begin{array}{l}
\text{loop} \\
\quad \text{if } c_i = 1 \wedge c_k = 1 \wedge c_z = 0 \\
\quad\quad z = i + k \\
\quad\quad c_i = 0 \\
\quad\quad c_k = 0 \\
\quad\quad c_z = 1
\end{array}
$$

A few words of explanation are in order. The statements occurring before the ''loop''statement in E^T are initial conditions; in accordance with the presentation in [Abadi90b], the environment is responsible for determining the initial conditions for all externally-visible state components. The ''if'' statements in these programs wait for their conditions to be true, and then proceed to the next statement. Each statement in the loop for E^T is executed by T's environment agents in the set $\neg\mu^T$, and each statement in M^T is executed by T's system agents in the set μ^T. Arbitrary amounts of stuttering are permissible between any two agent steps. The statement $k = ?$ indicates that state variable z may take on any value in its range.

The input-output synchronization scheme used here is as follows. An agent writing signal x waits for the clock line c_x to have the value 0 before setting x. Once x is set, c_x is changed to 1. An agent wishing to read signal x waits for the clock line c_x to have the value 1, reads x, and then sets c_x to 0 to indicate that the value has been read. If some such clocking scheme were not enforced, the set of possible behaviours for the composite system would be much larger, no simple relationships would exist among the sequences of states exhibited by T and D, and analysis of the security functionality of the composite system would be infeasible.

The above programs correspond to figure 9.1 in the obvious way. The environment for T writes input i, then writes input k, then reads output z, and goes back to writing i again, ensuring at each operation that the system is ready. Similarly, M^T waits for inputs i and k to be set, and for any previous value sent to output z to be read, sets z, fixes up the clock lines, and waits for more input.

We now digress to show that the above specifications, in program form, can easily be expressed in logical clause form. We also discuss liveness considerations which are more conveniently expressed in logical clause form.

First, we introduce an auxiliary state variable, the *program counter*, denoted here by *pc* (this is explained more fully in [Lamport89] and [Lamport90]). For example, here is M^T in logical clause form:

$$M^T = (pc = 1 \wedge \Box [N_{M^T}]_{\mu^T}),$$

where

$$\begin{aligned} N_{M^T} = {} & (pc = 1 \wedge c_i = 1 \wedge c_k = 1 \wedge c_z = 0 \wedge pc' = 2) \\ & \vee (pc = 2 \wedge z' = i + k \wedge pc' = 3) \\ & \vee (pc = 3 \wedge c'_i = 0 \wedge pc' = 4) \\ & \vee (pc = 4 \wedge c'_k = 0 \wedge pc' = 5) \\ & \vee (pc = 5 \wedge c'_z = 1 \wedge pc' = 1) . \end{aligned}$$

The formula N_{M^T} describes the *next-state relation* for M^T. A next-state relation is simply an *action*, a relationship that must hold among state variables across a behaviour step (in this case we are actually describing several possible steps in a single action). The formula M^T given above states what agents in μ^T are allowed to do, but does not require that they in fact do anything at all. To rule out such trivial implementations, we would normally augment M^T with some type of *liveness* requirement. The particular liveness condition chosen is of no consequence from a confidentiality viewpoint, but we will outline a simple condition in order to give a complete example.

One effective and general liveness condition is known as *strong fairness* [Lamport90], and can be defined on an action A as:

$$SF(A)_\mu \triangleq \Box\blacklozenge\ Enabled\ (A) \rightarrow \langle A \rangle_\mu ,$$

which means that, if A is enabled infinitely often, a behaviour step satisfying A must eventually occur. This allows for scenarios in which some system action is enabled, but then disabled (by some environment action) before the system has a chance to act. Only if the action is infinitely often enabled, is the system required to complete the action.

Adopting the strong fairness criterion as our liveness condition, M^T becomes:

$$M^T = (pc = 1 \wedge \Box [N_{M^T}]_{\mu^T} \wedge SF(N_{M^T})_{\mu^T}) .$$

This states that T starts with the first disjunctive clause in N_{M^T} partially enabled ($pc = 1$), obeys N_{M^T} on every step, and will actually take steps from N_{M^T} provided the environment behaves reasonably.

We will assume from here on that appropriate liveness requirements are part of the specification, and refrain from mentioning them further. The reader should now be convinced that the program form of specification is equivalent to the logical clause form, and can likely appreciate our preference for the program form. We now complete the specification of our example composite system.

The specification for system D is only slightly more complex than that for T:

$$D = (E^D \Rightarrow M^D) \ ,$$

where E^D is described by:

```
c_z = 0
c_l = 0
c_k = 0
loop
   if c_z = 0
      z = ?
      c_z = 1
   if c_l = 1
      c_l = 0
   if c_k = 1
      c_k = 0
```

and M^D is described by:

```
loop
   if c_k = 0
      k = ?
      c_k = 1
   if c_z = 1 ∧ c_l = 0
      l = k + z
      c_z = 0
      c_l = 1
```

By composing T and D we intend to use them together in some environment E such that:

$$(E^T \Rightarrow M^T) \cap (E^D \Rightarrow M^D) \subseteq (E \Rightarrow M^T \cap M^D) \ .$$

A suitable specification for this *global* environment E is:

$$
\begin{array}{l}
c_i = 0 \\
c_z = 0 \\
c_l = 0 \\
c_k = 0 \\
\text{loop} \\
\quad \text{if } c_i = 0 \\
\qquad i = ? \\
\qquad c_i = 1 \\
\quad \text{if } c_l = 1 \\
\qquad c_l = 0
\end{array}
$$

where the agent set for E is composed of agents neither in μ^T nor μ^D, that is, $\neg(\mu^T \cup \mu^D)$.

The specification for E is simpler than either E^T or E^D because it merely has to write the composite system's single input i, and read its single output l. For the composite system to truly work, the remaining portion of each system's environment property must be satisfied by the other system. Formally, this is the requirement that:

$$E \cap M^D \subseteq E^T ,$$

and

$$E \cap M^T \subseteq E^D ,$$

which can be combined to yield:

$$E \cap M^T \cap M^D \subseteq E^T \cap E^D .$$

The intersection of properties results in all possible interleavings of their agent steps. Here is a portion of the property $E \cap M^T \cap M^D$, with the agent involved at each step noted to the right, and each step numbered for reference:

1.	$c_i = 0$	
2.	$c_z = 0$	
3.	$c_l = 0$	
4.	$c_k = 0$	
5.	loop	
6.	if $c_k = 0$	μ^D
7.	$k = ?$	μ^D
8.	$c_k = 1$	μ^D
9.	if $c_i = 0$	$\neg(\mu^T \cup \mu^D)$
10.	$i = ?$	$\neg(\mu^T \cup \mu^D)$
11.	$c_i = 1$	$\neg(\mu^T \cup \mu^D)$
12.	if $c_i = 1 \wedge c_k = 1 \wedge c_z = 0$	μ^T
13.	$z = i + k$	μ^T
14.	$c_i = 0$	μ^T
15.	$c_k = 0$	μ^T
16.	$c_z = 1$	μ^T
17.	if $c_z = 1 \wedge c_l = 0$	μ^D
18.	$l = k + z$	μ^D
19.	$c_z = 0$	μ^D
20.	$c_l = 1$	μ^D
21.	if $c_l = 1$	$\neg(\mu^T \cup \mu^D)$
22.	$c_l = 0$	$\neg(\mu^T \cup \mu^D)$

Note that this program effectively specifies only a portion of the property $E \cap M^T \cap M^D$, since many other interleavings of these steps are possible. For instance, steps 6 through 8 and 9 through 11 are independent and the work of different agent sets, so they may be interleaved in any arrangement preserving the orders 6, 7, 8 and 9, 10, 11. Therefore these steps might occur in order 6, 9, 10, 7, 11, 8, or 9, 6, 10, 7, 8, 11, etc. However, step 12 must follow steps 8 and 11, since it requires particular state values set in those steps.

In the interests of brevity, we will remove all ''if'' statements and handshaking steps from our programs from here on, since we are really interested in the sequences of values assumed by the signals i, k, l, and z. Thus, property $E \cap M^T \cap M^D$ can be fully characterized (if we obey the clock synchronization scheme but do not report the steps involving them) by the following two programs:

loop	
$k = ?$	μ^D
$i = ?$	$\neg(\mu^T \cup \mu^D)$
$z = i + k$	μ^T
$l = k + z$	μ^D

and

$$\begin{array}{ll} \text{loop} & \\ \quad i = ? & \neg(\mu^T \cup \mu^D) \\ \quad k = ? & \mu^D \\ \quad z = i + k & \mu^T \\ \quad l = k + z & \mu^D \end{array}$$

Similarly, property $E^T \cap E^D$ can be fully characterized by the following two programs:

$$\begin{array}{ll} \text{loop} & \\ \quad k = ? & \neg\mu^T \\ \quad i = ? & \neg\mu^T \\ \quad z = ? & \neg\mu^D \end{array}$$

and

$$\begin{array}{ll} \text{loop} & \\ \quad i = ? & \neg\mu^T \\ \quad k = ? & \neg\mu^T \\ \quad z = ? & \neg\mu^D \end{array}$$

Actually, there are two more possibilities for $E^T \cap E^D$, since z may be set first in the loop. However, this results in behaviours which, except for the assignment to z on the first step, are cyclic permutations of the behaviours generated by the above programs. Thus, including them in property $E^T \cap E^D$ increases its size only very slightly, and since we are trying to show that another property is a subset of this one, we can ignore those behaviours.

These programs specify all permissible sequences of values for the state variables that they mention. From these programs it can be verified that

$$E \cap M^T \cap M^D \subseteq E^T \cap E^D .$$

The first three steps in the $E \cap M^T \cap M^D$ programs clearly are a specialization of the steps in the $E^T \cap E^D$ programs. The fourth step in the $E \cap M^T \cap M^D$ programs involves a state variable that is not even mentioned in $E^T \cap E^D$. Therefore, the former two programs are clearly specializations of the latter two programs, as required. For an example of a more formal proof of this type of hypothesis see section 5.3.3 of [Abadi90b].

This all goes to show that the composite system behaves according to $E \Rightarrow M^T \cap M^D$, which can be rewritten as $\neg E \cup (E \cap M^T \cap M^D)$. In the case where the environment is behaving properly, the composite system thus obeys $E \cap M^T \cap M^D$, which we have just characterized above by two five-line programs. Let us assume that the composite system hides the signals k and z, and presents only signals i and l to the outside world. Let us further suppress the identities of T and D by grouping μ^T and μ^D and referring to the union simply as μ, and replacing $\neg(\mu^T \cup \mu^D)$ by $\neg\mu$. We can then replace those programs by the following:

$$\begin{array}{ll} \text{loop} & \\ \quad i = ? & \neg\mu \\ \quad l = k + i + k = i & \mu \end{array}$$

which shows that the output l is simply a copy of the previous input i. Since i is a high-level signal, and l low-level, this represents a breach of confidentiality, almost regardless of the particular technical definition of confidentiality applied.

This result is important since both T and D can easily be shown to belong to the class of non-deducible properties, ND. Thus, since $T \in \mathrm{ND}$, $D \in \mathrm{ND}$, and $T \cap D \notin \mathrm{ND}$, ND is not a *composable* class.

We have discovered that the main reason for the non-composability of ND and similar confidentiality models is that they do not protect high outputs. Simply grouping **HO** with **HI** in the ND formulation results in a composable model (as we will see), but is unworkable in practice since it disallows ''writing up''. This has been pointed out by McLean [McLean90], among others, who has noted that non-deducibility does not limit information flow, but rather information sharing. In a later section we investigate a new confidentiality definition, based on the Flow Model, that protects **HO**, but still captures the notion of one-way information flow.

9.3.6. Implementation and Composition

Given a confidential (say, ND-closed) system S, that is, $S \in \mathrm{ND}$, an implementation S' will not in general satisfy the confidentiality property, since it need only satisfy $S' \subseteq S$. Thus, arbitrary behaviours may be excluded which are required for the property to be ND-closed. This is a simple explication of the well-known fact that removing non-determinism from a design can compromise confidentiality. In a security context, therefore, we must take special care with terms like ''implementation''.

Likewise, the composition of two properties, P and Q, is simply $P \cap Q$. It is therefore clear that, in general, the composition of confidential specifications does not result in a confidential composite system. One of the great strengths of our formalism, we believe, is the immediate emergence of the result that, in general, security properties are not preserved by implementation or composition.

We can build a framework for reasoning about implementation and composition by discovering, for arbitrary types of properties, what *features* of those property types allow them, or do not allow them, to be preserved upon implementation or composition. We construct a hierarchy of system descriptions, as shown in figure 9.2.

Each level of description consists of components which are sets of the components at the next lower level. At the lowest level, individual system runs are represented by behaviours. One level up, properties describe the

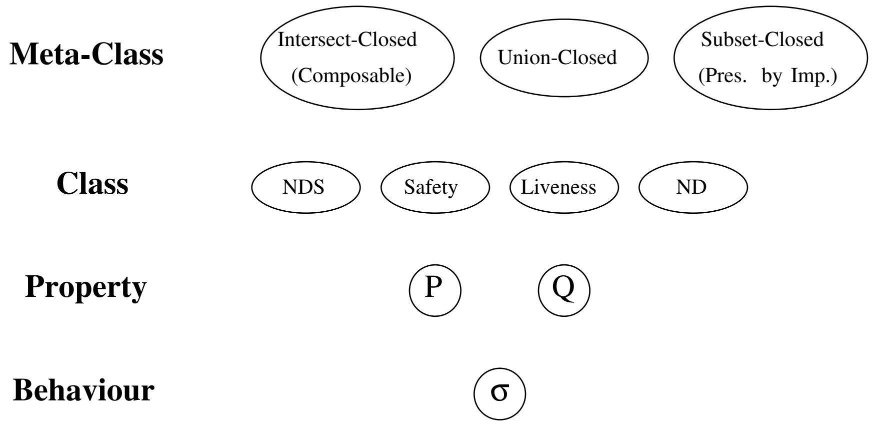


Figure 9.2: System Description Hierarchy.

activity of entire systems, or any portion thereof, by arbitrary stuttering-closed sets of behaviours. At the next level, *classes* are sets of properties that can be defined by arbitrary logical predicates, as we have demonstrated in defining, for example, the S, L, and ND classes. Components of the highest level, *meta-classes* , identify those classes which themselves satisfy some logical predicate. We have given three examples of meta-classes in figure 9.2 — classes that are closed under the intersection operation, and are thus *composable* ; classes that are closed under the subset operation, and are thus *preserved by implementation* ; and classes that are closed under the union operation.

9.4. THE POSSIBILISTIC FLOW MODEL

The flow model is a good candidate for study as a confidentiality property for two reasons. First, it has been shown [Gray91] that a system conforming to the flow model leaks no information from high to low levels, even in a statistical sense. Second, it has been shown [Gray91] that a system conforming to the flow model is also non-deducible on strategies, and non-deducibility on strategies has been shown [Millen90] to be composable. It should be pointed out that these features of FM have been proven in the context of system models which are quite different from ours. However, FM appears well-suited as a starting point for investigating composable confidentiality properties.

One of the strengths of the flow model is its elimination of statistical inference. The probabilistic state machine model which allows this, however, complicates the application and analysis of FM. Namely, systems to be analyzed must have explicit probabilities associated with each transition of the state machine.

We shall pursue here a modification of the flow model which is not based on a probabilistic state machine model, but preserves its composability and otherwise strong confidentiality. Below we derive this new class of confidentiality properties from the definition of the flow model, apply it to the example system from the previous section, and prove that it is composable. We then compare it to non-deducibility, and discover that it is strikingly similar in form, giving insight into the relationships between the comprehensive but complex probabilistic confidentiality properties, and the limited but simple non-deterministic models.

The central formula defining FM:

$$\mathbf{p}(L_s \cap H_s) > 0 \;\Rightarrow\; \mathbf{p}(L_t \mid L_s \cap H_s) = \mathbf{p}(L_t \mid L_s)\ ,$$

states that the probability of occurrence of any low-level state may depend on the preceding low-level state sequence, but not on the preceding high-level state sequence. Since the intention is to prevent low-level users from inferring anything about the history of the high-level state during an execution, it may be more intuitive to write the following formula:

$$\mathbf{p}(L_t \cap L_s) > 0 \;\Rightarrow\; \mathbf{p}(H_s \mid L_t \cap L_s) = \mathbf{p}(H_s \mid L_s)\ .$$

This can be read as, the probability of any high-level state sequence is unchanged by each successive low-level state, or equivalently, each new low-level state reveals no more about the preceding high-level state sequence than was revealed on the previous step. In fact, not only do both formulas express the same intent, that low-level state must be statistically independent from preceding high-level state, but they are algebraically equivalent.

We simplify the FM formulation by replacing the requirement for *equality* between probabilities with the requirement that probabilities must be both zero or non-zero. This is simplified further by the assumption that a behaviour set has non-zero probability if and only if it is non-empty. This assumption is reasonable since we have defined behaviours as *possible* executions of a system. Thus the requirement that $\mathbf{p}(X) = \mathbf{p}(Y)$, where X and Y are behaviour sets, is replaced by the requirement that $X \neq \emptyset \equiv Y \neq \emptyset$, and $\mathbf{p}(X) > 0$ is replaced by $X \neq \emptyset$.

Expanding the conditional probabilities from the FM formula gives

$$\mathbf{p}(L_s \cap H_s) > 0 \;\Rightarrow\; \frac{\mathbf{p}(L_t \cap L_s \cap H_s)}{\mathbf{p}(L_s \cap H_s)} = \frac{\mathbf{p}(L_t \cap L_s)}{\mathbf{p}(L_s)}$$

which requires equality between two ratios of behaviour set probabilities. Because intersection of behaviour sets can only result in smaller sets,

$$\mathbf{p}(L_s \cap H_s) > 0 \Rightarrow \mathbf{p}(L_s) > 0 \,,$$

which allows the formula to be simplified to

$$(L_s \cap H_s) \neq \varnothing \Rightarrow \left[(L_t \cap L_s) \neq \varnothing \equiv (L_t \cap L_s \cap H_s) \neq \varnothing \right] .$$

Again, because intersection of behaviour sets can only result in smaller sets,

$$(L_t \cap L_s \cap H_s) \neq \varnothing \Rightarrow (L_t \cap L_s) \neq \varnothing \,,$$

giving

$$(L_s \cap H_s) \neq \varnothing \Rightarrow \left[(L_t \cap L_s) \neq \varnothing \Rightarrow (L_t \cap L_s \cap H_s) \neq \varnothing \right] ,$$

which is equivalent to

$$(L_s \cap H_s) \neq \varnothing \wedge (L_t \cap L_s) \neq \varnothing \Rightarrow (L_t \cap L_s \cap H_s) \neq \varnothing \,.$$

This leads to the following definition.

We say that a property P is a *possibilistic flow-model* property if and only if it satisfies the following formula:

$$\forall t > 0 \quad \forall \alpha \in P \quad \forall \beta \in P \quad \forall \gamma \in P \quad :$$

$$(L_t \cap L_s) \neq \varnothing \wedge (L_s \cap H_s) \neq \varnothing \Rightarrow (L_t \cap L_s \cap H_s) \neq \varnothing \,,$$

where

$$L_t = \{ \sigma \in P \,:\, \mathbf{L}(\sigma_t) = \mathbf{L}(\alpha_t) \} \,,$$

$$L_s = \{ \tau \in P \,:\, \mathbf{L}(\tau_{0 \to t-1}) = \mathbf{L}(\beta_{0 \to t-1}) \} \,,$$

$$H_s = \{ \upsilon \in P \,:\, \mathbf{H}(\upsilon_{0 \to t-1}) = \mathbf{H}(\gamma_{0 \to t-1}) \} \,.$$

In words, given any low-level state at time t, L_t, any sequence of low-level states from time 0 to time $t-1$, L_s, and any sequence of high-level states from time 0 to time $t-1$, H_s, if L_t can follow L_s, and L_s can coexist with H_s, then all three must possibly coexist. If P is a possibilistic flow-model property, we write

$$P \in \mathtt{PFM} \,.$$

We can also rewrite the `PFM` definition without the need for identifying the L_t, L_s, and H_s behaviour sets:

$$\forall t > 0 \quad \forall \alpha \in P \quad \forall \beta \in P \quad \forall \gamma \in P \quad :$$

$$\exists \sigma \in P \,:\, \mathbf{L}(\sigma_t) = \mathbf{L}(\alpha_t) \ \wedge\ \mathbf{L}(\sigma_{0 \to t-1}) = \mathbf{L}(\beta_{0 \to t-1})$$

$$\wedge\ \exists \tau \in P \,:\, \mathbf{L}(\tau_{0 \to t-1}) = \mathbf{L}(\beta_{0 \to t-1}) \ \wedge\ \mathbf{H}(\tau_{0 \to t-1}) = \mathbf{H}(\gamma_{0 \to t-1})$$

$$\Rightarrow\ \exists \upsilon \in P \,:\, \mathbf{L}(\upsilon_t) = \mathbf{L}(\alpha_t) \ \wedge\ \mathbf{L}(\upsilon_{0 \to t-1}) = \mathbf{L}(\beta_{0 \to t-1}) \ \wedge\ \mathbf{H}(\upsilon_{0 \to t-1}) = \mathbf{H}(\gamma_{0 \to t-1}) \,.$$

Although the above two definitions correspond closely with the definition of `FM`, they are more complex than necessary. An equivalent formulation, which we shall adopt as the definition of PFM from this point forward is:

$$\forall t > 0 \quad \forall \alpha \in P \quad \forall \beta \in P \ :$$

$$Coexist\ (\mathbf{L}(\alpha_{0 \to t-1}), \mathbf{H}(\beta_{0 \to t-1})) \Rightarrow Coexist\ (\mathbf{L}(\alpha_{0 \to t}), \mathbf{H}(\beta_{0 \to t-1}))\ ,$$

where

$$Coexist\ (\mathbf{S}(\sigma_{0 \to j}), \mathbf{T}(\tau_{0 \to k})) \triangleq \exists \upsilon \in P : \mathbf{S}(\upsilon_{0 \to j}) = \mathbf{S}(\sigma_{0 \to j}) \wedge \mathbf{T}(\upsilon_{0 \to k}) = \mathbf{T}(\tau_{0 \to k})\ .$$

The *Coexist* predicate takes two partial state sequences, and returns true if a behaviour exists in P that supports both sequences. This formula requires that, if the first $t-1$ steps of some low-level state sequence are compatible with the first $t-1$ steps of any high-level state sequence, then the first t steps of that low-level state sequence must also be compatible.

Applying this definition to our example composite system, we see immediately that $T \in$ PFM, since T has only clock signals as low-level state. These signals operate completely indenpedently of the high-level state, and thus all legal low-level state sequences from T can coexist with any possible high-level state sequences. We will demonstrate in a more complete manner, however, that $D \notin$ PFM.

The ‘‘interesting’’ portion of D (that is, ignoring those behaviours which violate E^D) is $E^D \cap M^D$, and can be written in simplified program form as follows (clock signals can be ignored as for T, since they are independent of high-level state):

```
loop
   k = ?          μ^D
   z = ?          ¬μ^D
   l = k + z      μ^D
```

and

```
loop
   z = ?          ¬μ^D
   k = ?          μ^D
   l = k + z      μ^D
```

To show $D \notin$ PFM we must find one group of values for t, α, and β, that satisfies

$$Coexist\ (\mathbf{L}(\alpha_{0 \to t-1}), \mathbf{H}(\beta_{0 \to t-1}))\ ,$$

but does not satisfy

$$Coexist\ (\mathbf{L}(\alpha_{0 \to t}), \mathbf{H}(\beta_{0 \to t-1}))\ .$$

Figure 9.3 shows two such possibilities in a schematic form. Each diagram is a putative trace of l, k, and z over the first three or four steps. The diagrams also indicate the nature of the behaviour sets L_t, L_s, and H_s, in reference to the first definition of PFM above. The small box around the value of l at $t = 3$ indicates the nature of L_t: all behaviours in D which have $l = 1$ at $t = 3$.

Similarly, L_s is the set of behaviours which have $l = 0$ through their first three steps. In both diagrams, H_s is the set of behaviours which have $k = z = 0$ at $t = 0$, and $k = z = 1$ at $t = 2$; they differ in the order in which k and z change.

Now, the sequence of l values from $t = 0$ to $t = 2$ can coexist with the sequence of k and z values over the same period, since l need not change until after k and z have been set. However, $l = 1$ at $t = 3$ cannot occur, since l is the exclusive-or of k and z, even though, given some other high-level history, it could possibly follow the low-level history shown. Thus, $D \notin \mathtt{PFM}$.

It is possible to ''fix'' D so that $D \in \mathtt{PFM}$. The key is to conceal k until after l has been produced, thus preventing a Trojan horse from using k to affect future values of z. The improved version, D^*, is shown along with T in figure 9.4. The box labeled M is a memory latch that holds the value of the last internally-generated random bit, which is now renamed as variable j, and is not externally visible.

The specification for system D^* is

$$D^* = (E^{D^*} \Rightarrow \exists j : M^{D^*}) ,$$

---

t	l	k	z
0	0	0	0
	L_s	H_s	
1	0	1	0
2	0	1	1
3	1		
	L_t		

t	l	k	z
0	0	0	0
	L_s	H_s	
1	0	0	1
2	0	1	1
3	1		
	L_t		

Figure 9.3: Demonstrating $D \notin \mathtt{PFM}$.

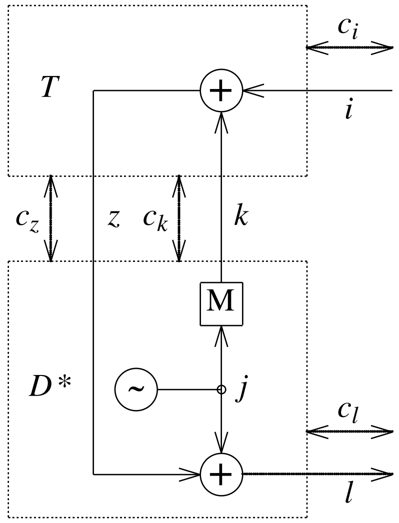


Figure 9.4: A Secure Composite System.

where $E^{D*} = E^{D}$, and M^{D*} is described by:

loop
    if $c_k = 0$
        $k = j$
        $c_k = 1$
    $j = ?$
    if $c_z = 1 \wedge c_l = 0$
        $l = j + z$
        $c_z = 0$
        $c_l = 1$

As was the case for D, the ‘‘interesting’’ portion of $D*$ is $E^{D*} \cap M^{D*}$, and can be written in simplified program form as follows

loop	
$z = ?$	$\neg \mu^{D*}$
$k = j$	μ^{D*}
$j = ?$	μ^{D*}
$l = j + z$	μ^{D*}

and

$$\begin{array}{ll} \text{loop} & \\ \quad k = j & \mu^{D*} \\ \quad z = ? & \neg\mu^{D*} \\ \quad j = ? & \mu^{D*} \\ \quad l = j + z & \mu^{D*} \end{array}$$

and

$$\begin{array}{ll} \text{loop} & \\ \quad k = j & \mu^{D*} \\ \quad j = ? & \mu^{D*} \\ \quad z = ? & \neg\mu^{D*} \\ \quad l = j + z & \mu^{D*} \end{array}$$

We can now prove that system $D*$ interacts with its environment in a secure manner.

Theorem:

$$E^{D*} \cap M^{D*} \in \texttt{PFM}.$$

Proof: We assume *Coexist* $(\mathbf{L}(\alpha_{0\to t-1}), \mathbf{H}(\beta_{0\to t-1}))$. There are two cases:

i) $\mathbf{L}(\alpha_t) = \mathbf{L}(\alpha_{t-1})$.
In this case the stuttering-closure of $E^{D*} \cap M^{D*}$ ensures *Coexist* $(\mathbf{L}(\alpha_{0\to t}), \mathbf{H}(\beta_{0\to t-1}))$.

ii) $\mathbf{L}(\alpha_t) \neq \mathbf{L}(\alpha_{t-1})$.
This occurs on the μ^{D*}-step that sets $l = j + z$. Recalling that l, j, and z are one-bit signals, the operator $+$ is a one-bit add (exclusive-or), and j is neither visible nor related in value to z, we see that $\mathbf{L}(\alpha_t)$ can take on any value, regardless of previous high state. Therefore, *Coexist* $(\mathbf{L}(\alpha_{0\to t}), \mathbf{H}(\beta_{0\to t-1}))$. $\square$

9.4.1. Composability of the Possibilistic Flow Model

The need for composable models of security becomes obvious once we recognize that any *use* of a system involves composition. This is because the user of a system takes the role of that system's environment. We have seen that specifications are written in the form $E \Rightarrow M$, but this is largely for the convenience of the system designer. When the system and environment interact, the resulting behaviours are in $E \cap M$, where the agents in E and M are disjoint, that is, $\mu^E \cap \mu^M = \emptyset$. If we want to guarantee that these behaviours are secure, therefore, we must ensure that both E and M are composably secure.

The composition of multiple machines, say $M_1, \cdots, M_n$, thus requires a proof that each M_i is composably secure. If we also specify an environment

E, it too must be proven composably secure. This supports our intuition that a system must be *used* securely if we expect to call it secure. For example, in discussing the confidentiality attributes of a system with high outputs and low inputs, we generally assume that the user of the system will not simply connect the high output back to the low input. We merely formalize this assumption by requiring that any machine, or, more abstractly, any environment, also be composably secure.

Our proof of the composability of PFM relies on the disjointedness of the components' agent sets. This ensures that the properties are independent, so that actions by an agent in one agent set cannot affect the membership of a behaviour in the other property. We say that a property P is "closed under $\neg\mu^P$-steps", a generalization of stuttering-closure.

Theorem:

If $P \in$ PFM, $Q \in$ PFM, and $\mu^P \cap \mu^Q = \varnothing$, then $P \cap Q \in$ PFM.

Proof:

We assume that t, $\alpha \in P \cap Q$ and $\beta \in P \cap Q$ are such that *Coexist* $(\mathbf{L}(\alpha_{0\to t-1}), \mathbf{H}(\beta_{0\to t-1}))$, that is,

$$\exists\, \sigma \in P \cap Q \;:\; \mathbf{L}(\sigma_{0\to t-1}) = \mathbf{L}(\alpha_{0\to t-1}) \;\wedge\; \mathbf{H}(\sigma_{0\to t-1}) = \mathbf{H}(\beta_{0\to t-1})\,.$$

We need to show that, for any α_t,

$$\exists\, \tau \in P \cap Q \;:\; \mathbf{L}(\tau_{0\to t}) = \mathbf{L}(\alpha_{0\to t}) \;\wedge\; \mathbf{H}(\tau_{0\to t-1}) = \mathbf{H}(\beta_{0\to t-1})\,. \tag{3}$$

There are two cases:

i) $\mathbf{L}(\alpha_t) = \mathbf{L}(\alpha_{t-1})$.
Construct $\tau \in P \cap Q$ such that $\tau_{0\to t-1} = \sigma_{0\to t-1}$, $\tau_t = \sigma_{t-1}$, and $\tau_{t+1\to\infty} = \sigma_{t\to\infty}$. τ satisfies (3), and must be in $P \cap Q$ because τ is stuttering-equivalent to σ, and $P \cap Q$ is stuttering-closed.

ii) $\mathbf{L}(\alpha_t) \neq \mathbf{L}(\alpha_{t-1})$.
This change must be due to an action by an agent in μ^P or μ^Q. Assume $\mathbf{a}(\alpha_t) \in \mu^P$. The proof is similar for $\mathbf{a}(\alpha_t) \in \mu^Q$. Since $P \in$ PFM,

$$\exists\, \upsilon \in P \;:\; \mathbf{L}(\upsilon_{0\to t}) = \mathbf{L}(\alpha_{0\to t}) \;\wedge\; \mathbf{H}(\upsilon_{0\to t-1}) = \mathbf{H}(\beta_{0\to t-1})\,.$$

Also, because $\mathbf{L}$ and $\mathbf{H}$ cover the externally visible state variables in P and Q, we know that $\upsilon_{0\to t-1} = \sigma_{0\to t-1}$. Since Q is closed under $\neg\mu^Q$-steps, there must be a behaviour $\tau \in P \cap Q$ such that $\tau_{0\to t} = \upsilon_{0\to t}$, satisfying (3). □

We can rewrite the non-deducibility definition in a form similar to PFM:

$$\forall\, t > 0 \quad \forall\, \alpha \in P \quad \forall\, \beta \in P \;:$$

$$\textit{Coexist}\,(\mathbf{L}(\alpha_{0\to t-1}), \mathbf{HI}(\beta_{0\to t-1})) \;\wedge\; (\mathbf{a}(\alpha_t) = \bot \;\vee\; \mathbf{a}(\beta_t) = \bot)$$

$$\Rightarrow\; \textit{Coexist}\,(\mathbf{L}(\alpha_{0\to t}), \mathbf{HI}(\beta_{0\to t}))\,.$$

This is different from PFM in three ways:

- ND ignores **HO**, under the pretext that high inputs alone need protection. We have argued that composability requires the protection of **HO**, even if high outputs are not autonomously generated.
- ND requires that the **L**-sequence and **HI**-sequence agree up to the tth step, whereas PFM only requires **L**-sequence agreement at the tth step. This is an attempt by PFM to reduce ND's information-sharing restriction to an information-flow requirement; since **HO** is protected, write-ups are allowed in one direction only (forward in time).
- As in the original ND formulation, we require the added clause that α and β not take simultaneous non-stuttering steps. Without this restriction, the definition is unsatisfiable by any non-trivial system.

With reference to the proof just presented on the composability of PFM, we can see that ND fails to be composable precisely because it makes no reference to **HO**. The proof fails for ND because **L** and **HI** do not cover the externally visible state variables in P and Q. PFM can now be seen as an intermediate model of confidentiality, with the non-probabilistic simplicity of ND, but the composability of FM. More work would be required to compare PFM with non-deducibility on strategies, and definitions of restrictiveness [McCullough88c] and forward correctability [Johnson88] are difficult to transcribe concisely into our formalism.

BIBLIOGRAPHY

[Abadi88] Abadi, M., and L. Lamport, ''The Existence of Refinement Mappings'', Research Report 29, Digital Equipment Corporation Systems Research Center, August 1988.

[Abadi90a] Abadi, M., ''An Axiomatization of Lamport's Temporal Logic of Actions'', Research Report 65, Digital Equipment Corporation Systems Research Center, October 1990.

[Abadi90b] Abadi, M., and L. Lamport, ''Composing Specifications'', Research Report 66, Digital Equipment Corporation Systems Research Center, October 1990.

[Aho72] Aho, A.V., and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Vol. 1, Prentice-Hall, 1972, p. 113

[Alpern85] Alpern, B., and F. Schneider, ''Defining Liveness'', *Information Processing Letters*, 21(4), October 1985, pp. 181-185.

[Bellare90] Bellare, M., S. Micali, R. Ostovsky, ''The (true) complexity of statistical zero knowledge'', *Proc. 22nd ACM Symp. on Theory of Computing*, 1990, pp. 494-502

[Bell74] Bell, D. Elliot, and Leonard J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, MTR-2547, MITRE Corp., Bedford, MA (March 1973).

[Bertalanffy68] Von Bertalanffy, L., *General System Theory. Foundations, Development, Applications.* New York: George Braziller, 1968

[Boebert85] Boebert, W.E., R.Y. Kain, ''A practical alternative to hierarchical integrity policies'', *Proc. 8th National Computer Security Conference*, Baltimore, Oct. 85, pp. 18-27.

[Burrows88] Burrows, M., M. Abadi, R. Needham, ''Authentication: A Practical Study in Belief and Action'', *Proc. 2nd Conf. on Theoretical Aspects of Reasoning about Knowledge*, IBM Almaden Research Center, Pacific Grove, CA. March, 1988, pp. 325-342.

[Chellas80] Chellas, B., *Modal Logic*. Cambridge University Press, 1980.

[Clark87] Clark, D.D., and Wilson, D.R., ''AComparison of Commercial aand Military Computer Security Policies'', *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, April 1987, pp.184-194.

[Clark88] Clark, D.D., and Wilson, D.R., ''Evolution of a model for computer integrity'', *Proc. 11th National Computer Security Conference*, Baltimore, Oct. 88, addendum.

[Cuppens90] Cuppens, F., ''An epistemic and deontic logic for reasoning about computer security'', *Proc. ESORICS 90*, AFCET, Toulouse, France, Oct. 1990, pp. 135-145.

[CTCPEC90] Canadian System Security Centre, *The Canadian Trusted Computer Product Evaluation Criteria* Version 2.0, Ottawa, Dec 1990.

[CTCPEC92] Canadian System Security Centre, *The Canadian Trusted Computer Product Evaluation Criteria* Version 3.0e, Ottawa, April 1992.

[DiVito88] DiVito, Ben, ''Derived Information Sensitivity as a Modelling Technique'', *Proceedings of the Security Foundations Workshop*, Franconia Inn, Franconia, NH, (June 1988)

[Fagin84] Fagin, R., J. Y. Halpern, M. Y. Vardi, ''A model-theoretic analysis of knowledge: preliminary report'', *Proc. 25th IEEE Symp. on Foundations of Computer Science*, 1984, pp. 268-278.

[Fagin86] Fagin, R., J. Y. Halpern, M. Y. Vardi, ''What can machines know? On the epistemic properties of machines'', *Proc. AAAI*, 1986, pp. 428-434.

[Glasgow88] Glasgow, J., G. MacEwen, and P. Panangaden, ''Reasoning about knowledge and permission in secure distributed systems'', *Proc. Workshop on Foundations of Computer Security*, Franconia, NH, 1988.

[Glasgow89] Glasgow, J., and G. MacEwen, ''Obligation as the basis of integrity specification'', *Proc. Computer Security Foundations Workshop II*, Franconia, NH, June 1989, pp. 64-70.

[Goguen82] Goguen, J.A., and J. Meseguer, ''Security Policies and Security Models'', *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, Oakland, Ca., pp. 11-20.

[Goguen84] Goguen, J. A., and J. Meseguer, ''Unwinding and Inference Control'', *Proceedings of the Symposium on Research in Security and Privacy*, IEEE Computer Society, April 1984, pp. 144-161.

[Goldwasser85] Goldwasser, S., S. Micali, and C. Rackoff, ''The knowledge complexity of interactive proof systems'', *1984 ACM Foundations of Computer Science*, ACM 1985, pp. 291-304.

[Gray90] Gray, J.W.III, ''Probabilistic Interference'', *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, Ca., pp. 170-179

[Gray90] Gray, R. M., *Source Coding Theory*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1990.

[Gray91] Gray, J. W., ''Toward a Mathematical Foundation for Information Flow Security'', *Proceedings of the Symposium on Research in Security and Privacy*, IEEE Computer Society, May 1991, pp. 21-34.

[Guttman88] Guttman, J. D., and M. E. Nadel, ''What Needs Securing?'', *Proceedings of the Computer Security Foundations Workshop*, IEEE Computer Society, June 1988, pp. 34-57. June 12-15 1988

[Havender68] Havender J.W., ''Avoiding deadlock in multitasking systems'', *IBM System Journal*, vol. 7 no. 2, 1968, pp. 74-84.

[Hintikka62] Hintikka, J., *Knowledge and belief*, Cornell University Press, 1962.

[Hoare85] Hoare, C.A.R., *Communicating sequential processes*, Prentice-Hall 1985.

[Hopcroft69] Hopcroft, J. E., and J. D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass. 1969.

[ISO7498-2] International Organization for Standardization, *OSI Security Architecture*, International Standard ISO 7498 Addendum 2.

[ISO9000] International Organization for Standardization, *Quality Management and Quality Assurance Standards*, International Standard ISO-9000.

[ITSEC89] German Information Security Agency, *Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems*, 1st Version, Bundesanzeiger, Köln, Germany, 1989.

[Jacob88] Jacob J., ''Security specifications'', *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE 1988, pp. 14-23.

[Johnson88] Johnson, Dale M., and F. Javier Thayer, ''Security and the composition of machines'', *Proceedings of the Security Foundations Workshop*, Franconia Inn, Franconia, NH, June 12-15 1988

[Jueneman89] Jueneman Robert R., ''Integrity controls for military and commercial applications'', *Proc. Invitational Workshop on Data Integrity*, NIST, Jan. 89, sect. A.5

[Klir85] Klir, G.J., *Architecture of Systems Problem Solving*, Plenum Press, New York, 1985

[Kripke63] Kripke, S.A., ''Semantical analysis of modal logic I, normal propositional calculi'', *Zeitschrift fuer mathematische Logik und Grundlagen der Mathematik*, VEB Deutscher Verlag der Wissenschaften, vol. 9 (1963), pp. 67-96.

[Kumar86] Kumar, P.R., and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control*, Prentice-Hall, Englewood Cliffs, NJ (1986)

[Lamport83] Lamport, L., ''Specifying Concurrent Program Modules'', *ACM Transactions on Programming Languages and Systems*, 5(2), April 1983, pp. 190-222.

[Lamport84] Lamport, L., ''Basic concepts'', *Advanced Course on Distributed Systems*, Lecture Notes on Computer Science 190, Berlin-Heidelberg: Springer, 1984

[Lamport89] Lamport, L., ''A Simple Approach to Specifying Concurrent Systems'', *Communications of the ACM*, 32(1), January 1989, pp. 32-45.

[Lamport90] Lamport, L., ''A Temporal Logic of Actions'', Research Report 57, Digital Equipment Corporation Systems Research Center, April 1990.

[Lee88] Lee, T.M.P., ''Using mandatory integrity to enforce commercial security'', *Proc. 1988 IEEE Symposium on Security and Privacy*, Oakland, Apr. 88, pp. 140-146.

[Lipner82] Lipner, S.B., ''Non-discretionary controls for commercial applications'', *Proc. 1982 IEEE Symposium on Security and Privacy*, Oakland, Apr. 82, pp. 2-10.

[Marcus86] Marcus, Leo, and T. Redmond, ''A Semantics of Read'', *Proc. 9th National Computer Security Conference*, Baltimore, MD, National Bureau of Standards, Sept. 1986, pp.184-193

[Marcus88] Marcus, Leo, and Timothy Redmond, ''A model-theoretic approach to specifying, verifying, and hooking up security policies'', *Proceedings IEEE Symposium on Security and Privacy*, IEEE 1988, pp.127-138.

[Marcus89] Marcus, Leo, ''The search for a unifying framework for computer security'', *Fall '89 CIPHER newsletter*, IEEE Computer Society Technical Committee on Security and Privacy, pp. 55-63.

[McCullough87] McCullough, Daryl, ''Specifications for multi-level security and a hook-up property'', *Proc. 1987 IEEE Symposium on Security and Privacy*, IEEE 1987.

[McCullough88a] McCullough, Daryl, *The Theory of Security*, Technical Report 11-12, Odyssey Research Associates Inc., Ithaca, NY, (1988).

[McCullough88b] McCullough, Daryl, ''Covert Channels and Degrees of Insecurity'', *Proceedings of the Workshop on Foundations of Computer Security*, Franconia, NH, (1988).

[McCullough88c] McCullough, Daryl, ''Noninterference and the composability of security properties'', *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE 1988, pp. 177-186.

[McLean90] McLean, J., ''Security Models and Information Flow'', *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, Ca., pp. 180-187.

[Mesarovic68] Mesarovic, M.J., ''On Some Mathematical Results As Properties of General Systems'' *Mathematical Systems Theory,* Vol 2, No. 3, 1968, pp. 203-222.

[Millen87] Millen, J. K., ''Covert Channel Capacity'', *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 60-66.

[Millen90] Millen, J., ''Hookup Security for Synchronous Machines'', *Proceedings of the Computer Security Foundations Workshop III*, IEEE Computer Society, June 1990, pp. 84-90.

[Moser89] Moser, L.E., ''Alogic of knowledge and belief for reasoning about computer systems'', *Proc. Computer Security Foundations Workshop II*, Franconia, NH, June 1989, pp. 57-63

[Nyquist24] Nyquist, H., ''Certain factors affecting telegraph speed'', *Bell Syst. Tech. Jour.*, vol. 3, April, 1924, p. 324.

[Pittelli88] Pittelli, Paul A., ''Formalizing integrity using non-interference'', *Proc. 11th National Computer Security Conference*, Baltimore, Oct. 88, pp. 38-42.

[Reiter87] Reiter, R., ''Nonmonotonic reasoning'', *Annual Review of Computer Science*, 1987, pp. 147-186

[Shannon49] Shannon, C. E., ''Communication in the Presence of Noise'', *Proc. of the IRE*, vol. 37, pp. 10-21, January, 1949. Reprinted in *Proc. of the IEEE*, vol. 72, no. 9, pp. 1192-1201, September 1984.

[Solovay77] Solovay, R., and V. Strassen, ''A fast Monte-Carlo test for primality'', *SIAM J. Computing*, vol. 6, no. 4, Dec 1977, pp. 84-85.

[Sutherland86] Sutherland, David, ''A Model of Information'', *Proc. 9th National Computer Security Conference*, National Bureau of Standards, September 1986, pp. 175-183.

[Sutherland89] Sutherland, Ian, S. Perlo, R. Varadarajan, ''Deducibility Security with Dynamic Level Assignment'', *Proceedings of the Second Workshop on Foundations of Computer Security*, Franconia, NH, (1989), pp. 3-8.

[TCSEC85] National Computer Security Center, *Department of Defense Trusted Computer Security Evaluation Criteria*, DOD 5200.28-STD, Dec 1985.

[Thomson88] Thomson, Brian, E.S.Lee, P.I.P. Boulton, M. Stumm, D.M. Lewis, *A Trusted Network Architecture*, Technical Report CSRI-228, Computer Systems Research Institute, University of Toronto, Toronto, Ontario (Oct. 1988)

[Thomson89] Thomson, Brian, E.S.Lee, P.I.P. Boulton, M.Stumm, D.M.Lewis,. ''Deducibility-based Network Security Modelling'', *Proceedings of the First Canadian Computer Security Conference*, Ottawa (1989)

[TNI87] National Computer Security Center, *Trusted Network Interpretation of the Trusted Computer Security Evaluation Criteria*, NCSC-TG-005, July 1987.

[Williams88] Williams, John C., and M. L. Day, ''Sensitivity Labels and Security Profiles'', *Proceedings of the 11th National Computer Security Conference*, National Bureau of Standards, October 1988, pp. 257-266.

[Wittbold90] Wittbold, J.T., and D.M. Johnson, ''Information Flow in Nondeterministic Systems'', *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, Ca., pp. 144-161

[Yu88] Yu, C.-F., and V.D. Gligor, ''A formal specification and verification method for the prevention of denial of service'', *Proc. 1988 IEEE Symposium on Security and Privacy*, IEEE 1988, pp. 187-202.