Computers & Security, 10 (1991) 10 (1991) 443–465

Operating System Integrity

G. F. G. O'Shea

Information Technology Security Group, KPMG Peat Marwick McLintock, P.O. Box 486, 1 Puddle Dock, London EC4V 3PD, and Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HU, U.K.

Although the issue of integrity in operating systems is generally perceived to be of fundamental importance in the preservation of security in a computer system, the term is commonly used to capture a variety of different notions. This paper considers and compares different notions of integrity that have been applied in the context of operating systems, and discusses their implications for the overall design and function of operating systems. Several seminal proposals aimed at improving integrity controls are reviewed, and their relation to current developments and contemporary practices is discussed.

Keywords: Operating systems, Integrity, Security, Access control

1. Introduction

The issue of operating system integrity is widely considered to be of considerable importance to the control and security of computer systems, but it is often far from clear what is intended by the term "integrity", which is frequently used in the context of computer systems to capture such notions as quality, correctness, prevention of unauthorised modification, protection and appropriateness. It is frequently described as one of three aspects of "security", namely confidentiality, integrity and availability [7]. In the absence of a clear definition it is difficult to conduct any serious discussions or to specify systems with the property of "integrity".

The confusion that this can cause is analogous to that which often results when people attempt to discuss "security" without defining what they mean by the term. This confusion can obscure matters of real concern, leaving potential or real problems poorly resolved.

This paper does not attempt to cover integrity issues that might be applied in a wider context, many of which are quite distant from the services and controls that an operating system can provide (for example, internal database integrity or specific application related issues).

Many of these higher level interpretations of integrity, which we will refer to as "data integrity", are concerned with the integrity of data stored within a system as perceived by a user of an application running in the system. The requirement is that data are manipulated strictly in accordance with user expectations by application programs which preserve certain properties of the data such as consistency between related data items. These issues are often application specific. The desired properties of the data are assured as a result of confidence in the specification and correctness of the application programs, which is valid only if the data are not altered by other means.

Operating system features are often too primitive to address precise data integrity requirements directly, but they are often expected to provide mechanisms that support the controls deployed in achieving data integrity. To provide such controls an operating system must be resistant to deliberate attempts to subvert the controls it provides. This

0167-4048/91/$3.50

paper explores various interpretations of these requirements, and controls that have been devised to meet them.

To avoid unnecessary confusion, this paper distinguishes between security and integrity, and does not consider integrity merely as one aspect of security. It uses the term security to refer to properties and mechanisms concerned with preserving the confidentiality of information according to some defined policy. This is consistent with use of the term "security" in much of the operating system literature, which has often been concerned with military requirements. The paper concludes by offering an interpretation of integrity issues in operating systems which is derived from the main themes of the reviewed material.

The views expressed in this paper are those of the author, and do not necessarily express any opinion of any of the organisations to which he is affiliated.

2. Interpretations of Integrity

Different interpretations of integrity and other work of significance are presented in the main body of the paper. With the exception of the first, they are presented in roughly chronological order.

2.1 The Trusted Computer System Evaluation Criteria

The Trusted Computer System Evaluation Criteria (referred to hereafter as the TCSEC) is published by the United States Department of Defense Computer Security Center (now called the National Computer Security Center) [5]. It was produced following an extensive programme of research into computer security requirements for military and government systems, with the intention of encouraging commercial development of systems which would meet those requirements (at a reasonable cost). An evaluation scheme was established to certify products claiming to meet various levels of the criteria.

It is difficult to discuss any aspect of computer security without referring to the TCSEC. It is currently the major stimulus leading to improvement of security in contemporary computer operating systems, with every indication that this will continue to be the case.

The TCSEC identifies a hierarchy of seven different evaluation classes. The different classes are distinguished by functional and assurance requirements, both of which increase from one class to the next.

A fundamental concept at all levels is that of the "reference monitor", which is required to enforce authorised access relationships between the subjects and objects that form the system. Implementations of the reference monitor concept is known as a "reference validation mechanism" or "security kernel", and must satisfy three requirements.

(1) It must be tamper proof.

(2) It must always be invoked.

(3) It must be small enough for thorough and complete analysis and testing to be performed.

Most contemporary operating systems fail to meet the third of these requirements, since they are often large and highly complex. This issue is essentially one of assurance that the system can be relied upon to do what is required of it, in this case reflected in design requirements. Other assurance issues, such as the amount of testing and the development techniques to be used, depend on the evaluation class in question. The TCSEC therefore introduces the concept of a Trusted Computing Base or TCB to describe that part of such systems which implements the reference monitor functional requirements, with the protection boundary and contents of the TCB (which may be an entire operating system) providing the basis for evaluation testing.

The following summary gives some flavour of the evaluation classes.

D: Minimal Protection (*i.e.* failed to meet one of the higher classes).

C1: Discretionary Security Protection, suitable for a cooperating community of users using data of a single security classification.

C2: Controlled Access Protection, able to enforce individual accountability for use actions through finely grained access control and audit trails.

B1: Labelled Security Protection, able to enforce a "mandatory" security policy over parts of the system, so that user actions can be constrained even for data that they control.

B2: Structured Protection, extending the mandatory controls over the entire system, and requiring careful separation of security critical components within its TCB.

B3: Security Domains, requiring that non-essential components (from a security perspective) be excluded from its TCB, which must be amenable to thorough analysis and testing.

A1: Verified Design, functionally similar to class B3, but requiring the use of formal software engineering techniques in its development.

The few contemporary operating systems that have achieved certification tend to be limited to classes C1 and C2, setting themselves the goal of B1 certification.

The significant difference between classes C2 and B1 is the introduction of mandatory access controls, based upon "sensitivity labels" associated with the subjects and objects in the system. Each sensitivity label has two components. The first is a "hierarchic classification" selected from an ordered set of eight or more labels (this paper will refer to these as "levels"). The second component is a set of "non-hierarchic categories", selected as a subset of categories defined for the system. The principle is that all subjects (active entities, such as processes) and objects (passive entities, such as files) are marked with appropriate labels, and that the system will use these as the basis of access control, overriding any discretionary access controls that may also be in use.

The mandatory access controls are required to enforce a security policy known as a Multi-level Security (or MLS).

For a subject to read an object, its security level must be greater than or equal to that of the object, and its security categories must contain (be a superset of) all those of the object. We can express this in an obvious way as:

$$\text{level(s)} \geqslant \text{level(o)} \wedge \{\text{subject categories}\} \supseteq \{\text{object categories}\}$$

For a subject to write to an object, its security level must be less than or equal to that of the object, and its security classifications must be no greater than those of the object (*i.e.* they are subset of those of the object). We can express this as:

$$\text{level(s)} \leqslant \text{level(s)} \wedge \{\text{subject categories}\} \subseteq \{\text{object categories}\}$$

The objective of the mandatory access controls is to prevent highly sensitive information from being accessed or leaked to individuals not cleared for it, which is done by associating labels with such objects as files and users at the time they log in.

The MLS controls are based upon the Bell and Lapadula model (a formal specification) of the desired behaviour of a computer system enforcing this policy [2]. The Bell and Lapadula model describes a state machine which enforces the desired policy over all sequences of state transitions, provided only that it starts in a valid state and that each individual state transition preserves the desired policy. This makes it exceptionally useful when attempting to construct a system with a high degree of assurance, since it simplifies the program verification effort.

The TCSEC "read" controls are a direct interpretation of the Bell and Lapadula "simple security" property, and its "write" controls a direct inter-

pretation of their "star" property (usually written *-property). The intention of the former is obvious, since it prevents a subject from reading information it has not been cleared for. The *-property is considered to be less obvious, but prevents a subject from leaking sensitive information to others not cleared for it (*e.g.* by writing secret information into a public file).

Integrity issues in the TCSEC are thus concerned with correctness and protection of the TCB, and with the definition and separation of objects under its control. It provides some support for data integrity requirements in its discretionary access controls, but these are crude and are not emphasised or functionally enhanced at higher evaluation levels.

2.2 Biba's Models of Integrity

One of the earliest attempts to build a highly secure system was the Kernel Multics project in the late 1970s [21]. This project included a study by Biba of integrity issues in the context of the MLS mandatory access mechanism for controlling information confidentiality [3].

Biba observed that MLS policies are only concerned with controlling dissemination of information, while the validity of information in a computer system requires control over modifications made to information. He called this latter issue "integrity", and studied various requirements, policies and mechanisms that could be applied to achieving integrity in computer systems and the Multics operating system kernel in particular.

Biba informally defined integrity in a computer system to be a guarantee that the system performs as intended by its creator, with properties of the behaviour specified by its creator being beyond the scope of study. Much of this is concerned with the quality of software engineering employed during the development of the system (which is the point of the "assurance" levels of the TCSEC). The aspect of integrity that Biba specifically studied was the deployment of access control mechanisms to protect well-engineered systems reliably against malicious modification that could subvert their integrity by causing them to behave in a way not intended by their creator.

Biba classified the threats to the integrity of a programmed subsystem (some part of the complete system) under the following headings.

- External direct: where one subsystem directly modifies code or data belonging to another, *e.g.* where parameter validation performed in calls to a security kernel is incomplete or can be otherwise compromised.

- External indirect: *e.g.* using a Trojan horse to cause another subsystem to perform some unintended action.

- Internal direct: *e.g.* self-modifying code.

- Internal indirect, *e.g.* unintentional self-modification of code.

He also classified the mechanisms that could be used in preserving integrity, using as his criteria.

- Static: performed once only, essentially through the use of program verification techniques, so that code is known to perform only desirable actions in advance.

- Dynamic: those that provide run-time enforcement of integrity policies.

- Grain: for example, noting that mechanisms supporting access control at the grain of operating system files cannot enforce a policy that requires separation of sets of records within a single file.

Biba considered three instances of integrity requirements in the context of secure military systems, as follows.

- National Security. Recognizing that correct functioning of a system may depend on the integ-

rity of objects that are widely available under MLS policy (*e.g.* library subroutines), he observed that a single access control level cannot be used to express both policies (*e.g.* readable by all, modifiable by none). He also noted that similar considerations apply outside of the military domain.

- User Identity: essentially the deployment of discretionary access controls granting the right to modify an object based on the identity of the requesting user.

- Protected Subsystems: programmed subsystems that may be invoked by external subjects to perform operations they are not allowed to perform themselves. The subsystem must be protected from subversion attempts by external subjects, and the external subjects themselves may require protection against possibly undesirable actions by the subsystem. A common example of a protected subsystem is the TCB or security kernel of an operating system.

Biba described several integrity policies that could be applied in meeting the above requirements. Some of these were discretionary in nature, based on access control lists, which suffer from the same issues of reliability (through complexity and the inability convincingly to demonstrate support for a particular policy) that limit their usefulness in supporting security policies. Of greater interest are his mandatory integrity policies, which are considered below.

Biba's mandatory policy models describe a system in terms of the objects and subjects it contains, all of which are marked with integrity labels consisting of an integrity level and a set of integrity categories (analogous to those used in MLS). The notation

$$I(s) \leqslant I(o)$$

means "the integrity label of subject s is less than or equal to the integrity label of object o".

Biba suggested that integrity and security levels are likely to be disjoint, but that integrity and security category sets may have elements in common, since they are largely concerned with distinctions between functional areas. He considered that the assignment of integrity labels to users (more accurately, to subjects operating on their behalf) would be derived using much the same approach as is used for security labels, *i.e.* the trustworthiness of the individual. The assignment of integrity labels to objects is performed so as to achieve control over the modification of objects, and is fundamentally different from the objective when assigning security labels.

The principle involved is that a subject may

- read objects of greater or equal integrity;

- write to objects of lower or equal integrity;

- invoke other subjects of a lower or equal integrity (since the state of the invoked subject is "modified" as a result, with procedure call and return being treated as two separate "invoke" operations).

Biba described a number of integrity policies which could be based on derivatives of this basic foundation.

The first variant of the Low Watermark policy allows the integrity of a subject to decrease as a result of reading objects with a lower integrity, so that the integrity of the subject cannot exceed the integrity of any object it has read. The integrity write rule is not relaxed, so that problems can occur if a file is read with a lower integrity than a file the subject has open for write access (such files would suddenly "disappear" from the subject).

The second variant of the Low Watermark policy decreases the integrity of an object to that of any subject which has written to it, but has the problem that objects generally tend to decrease in integrity level. Notably, this policy does not prevent

improper modifications, but helps to make them apparent.

The ring policy allows subjects to read objects of any integrity, but only to modify those of a lower or equal integrity level. This results in the subject assuming responsibility for maintaining the integrity of the modified object if it has read information of lower integrity.

The strict integrity policy has fixed integrity labels for both subjects and objects. This can tend to make many objects within the system unreadable to a given subject (because its integrity level is greater than that of the objects), but it also prevents calls to programs of lower integrity (since these cannot be read either).

Biba recognised that the composition of a security lattice and an integrity lattice itself forms a lattice. He described the effects of such a lattice in terms of the ability to read or write to an object, since these operations are constrained under both the security and integrity policies. The result is shown in Table 1, and its most noticeable feature is the strict limitation placed on the ability both to read and to write to an object.

Biba recognised that many integrity issues are application specific, which placed them beyond the scope of his study. His objective was to identify a mandatory policy which would enforce integrity control commensurate with the appropriateness of the labels assigned.

TABLE 1 Biba's observation concerning read (R) and write (W) access constraints when the composition of an MLS security lattice and his integrity lattice is enforced. The rows show the relation between integrity levels of subjects (s) and objects (o), while the columns show the relation between their security levels. The matrix entries show whether a subject is permitted read (R) or write (W) access to an object under the composition of the integrity and security constraints

Integrity level	Security level		
	$S(s) < S(o)$	$S(s) = S(o)$	$S(o) < S(s)$
$I(s) < I(o)$		R	R
$I(s) = I(o)$	W	W,R	R
$I(o) < I(s)$	W	W	

He suggested that the strict integrity policy appeared to be the most appropriate for use in the military environment, and that it would be most beneficial in situations where a significant amount of cross-integrity sharing of objects and information is involved. A particular benefit he attributed to the strict policy was its simplicity and the fact that it is universally applied, which makes it easier to enforce certifiably. He also observed that existing verification methodologies, intended for verification of a security kernel against an MLS security policy, work for any partially ordered set of labels. Thus the available tools and technology could be used in the production of security kernels requiring high assurance of their ability to enforce a mandatory integrity policy.

He suggested that the Low Watermark policy would be undesirable in practice, but remarked that the utility of any integrity policy would have to be justified by its use.

The Biba models stressed the importance of correctness in notions of integrity and captured a class of integrity policies based on information flow. The correctness aspects of a high integrity system will be in respect of the system specification, and if the specification is at application level then a large degree of data integrity will result. The Biba models of integrity exclude application issues, concentrating on properties of a system that would enhance assurances of integrity for the TCB and for classes of data stored within the system. Because the strict integrity model can be used to effectively partition a system, it can be used to constrain the damage potential of incorrect or malevolent programs such as computer viruses [25].

2.3 IBM MVS and VM/SP Operating Systems

IBM has made statements of "system integrity" for its VM and MVS families of operating system [9,

10]. We will consider only the MVS case, since they are similar in intent.

For MVS, this is defined as the provision of mechanisms under the customer's control which can prevent an unauthorised program from:

- circumventing or disabling memory protection mechanisms;

- avoiding password or access control mechanisms (referring specifically to IBM's Resource Access Control Facility product);

- obtaining supervisor state, a privileged memory protection key value or "APF" authorisation (from which state most other forms of privilege can be obtained).

The issue of integrity in MVS is the major theme of a set of guidelines that IBM makes to its customers [12] concerning issues that must be considered when designing and implementing modifications or extensions to the MVS operating system. This practice is widespread among users of MVS, and has been responsible for the introduction of significant flaws on many occasions [19, 20]. MVS "integrity support" restricts only unauthorised (non-privileged) programs, and IBM holds the customer responsible for access controls on libraries containing authorised programs and the effects of any authorised programs they develop (which may take the form of user-written system call handlers or "user-exits", amongst others).

IBM identifies a number of potential flaws that a customer may introduce via authorised code, and offers programming conventions that address these issues, although the "conventional" technique of taking a local copy of the parameters is not mentioned.

- To prevent an unauthorised program from obtaining inappropriate access to storage areas not directly accessible to it, the authorised code should run with the storage protection key of the caller when using addresses supplied to it as parameters.

- When an unauthorised program passes an address that is supposed to refer to an MVS data structure, the authorised code should validate it by following a chain of data structures from some known origin.

- Authorised code must retain sufficient information concerning resources under its control to prevent unauthorised programs from gaining inappropriate access to a resource with a similar name to one it has legitimate access to (typically files with the same name but residing on different disks).

- When a system call handler is invoked by an unauthorised program, and itself causes a different system call to occur, there is a possibility that the second system call handler will perform only limited parameter validation when it detects it is being called by an authorised program. The recommended approach is for the first system call handler to assume the protection key of its unauthorised caller before invoking the second system call. The second system call handler should then perform full parameter validation.

These amount to a particularly narrow view of integrity. It is concerned primarily with the TCB, and in particular with the requirements that it be tamper-proof, that any access control mechanisms it may provide cannot be circumvented and that it always be invoked. To meet these requirements it is necessary to arrange that the TCB be protected and that access to resources under TCB control can only be made through the TCB. Memory protection and supervisor state are the protection primitives used for this purpose in the System 370 architecture. The requirement concerning access control mechanisms makes provision for the use of packages such as RACF, ACF2 and Top Secret which enhance the function of the MVS TCB to meet TCSEC level C2. Without such products

MVS provides only crude access control mechanisms.

2.4 Lipner's "commercial" applications

Lipner produced a paper in 1982 that described the use of the lattice-based mandatory access controls in meeting commercial security requirements without mandating a particular security policy [18].

Lipner suggested that preventing unauthorised modification of data is probably more important in the majority of commercial installations than preserving its confidentiality, so he sought an example of commercial security requirements on which to base his discussion. Following consultation with a senior EDP auditor, he established the following security requirements as representative of commercial policy.

(1) Production users of the system operate on production data using production programs, and cannot write programs of their own.

(2) Development staff use test versions of programs and data, but cannot access the production versions. Copies of production objects may be obtained when necessary through a special process.

(3) Promotion of programs from the development to the production environment is a controlled event.

(4) The activity of system programmers must be controlled and recorded.

(5) Management and auditors are permitted to interrogate the system state and audit trails.

Lipner demonstrated that label-based integrity controls designed to support multi-level security can be used to enforce this policy, although the semantics of the labels and constraints are different. The approach he described is illustrated in Table 2. It depends upon users not being able to select a subset of their clearance levels for use in a particular session, and Lipner accepts that it lacks refine-

TABLE2 Lipner's scheme for enforcing "commercial" integrity requirements using a conventional MLS lattice.[a] Objects and subjects are shown with their security label values for security level and security categories, and the matrix cells show whether a subject can read (R) or write (W) to the corresponding object

	Prod. data (SL,{PD,PC})	*Prod. code (SL,{PC})*	*Dev. app. (SL,{D,T})*	*Dev. sys. (SL,{SD,T})*	*Tools (SL,{T})*	*Sys. pgm. (SL,{ })*	*Audit trail (AM,{any})*
System management and audit (AM, {any})	R	R	R	R	R	R	RW
Production users (SL, {PD,PC})	RW	R				R	W
Application programmers (SL, {D,T})			RW		R	R	W
System programmers (SL, {SD,T})				RW	R	R	W
System control (SL, {all})	RW	RW	RW	RW	RW	RW	W

[a]The labels are interpreted thus: SL < AM; PD = Production Data; PC = Production Code; SD = System Development; T = Software Tools; D = Development; SL = System-Low; AM = Audit-Manager.

ment (using only two security levels and four security categories). In principle it uses the Bell and Lapadula simple security property to control reading of objects, and the *-property to control writing of objects through the introduction of additional labels such as T (software tools) and PC (production code).

Lipner demonstrated that a composition of the Bell and Lapadula security lattice and the Biba integrity lattice can be used to enforce these requirements in a simple and more appealing way (Table 3). This scheme dispenses with the T and PC security categories, achieving equivalent control using integrity labels, and in particular the use of integrity levels to protect software against modification.

Notable features of the composed lattice are the use of the "operational" integrity level to prevent modification of production software (since no users can log on with operational or system-program integrity levels) and the introduction of "repair" users and "repair" software. This latter feature allows an additional requirement to be enforced, so that repair activity on production databases can only be performed by special-purpose software. In Lipner's scheme, repair software must have the same labels as production users, and repair users the same labels as production users so that they can operate on production data. There is therefore a need to use discretionary access control mechanisms or to perform relabelling when repair software is used.

TABLE 3 Lipner's scheme employing the composition of an MLS security lattice and a Biba integrity lattice.[a] Access to objects by subjects is constrained by security labels and integrity labels (shown in that order). Security and integrity labels are ordered so that SL < AM and SL < O < SP

	Prod. data (SL, {P}) (SL, {P})	*Prod. code* (SL, {P}) (O, {P})	*Dev. app.* (SL, {D}) (SL, {D})	*Dev. sys.* (SL, {SD}) (SL, {D})	*S/W tools* (SL, { }) (O, {D})	*Sys. pgm.* (SL, { }) (SP, {P,D})	*Repair code* (SL, {P}) (SL, {P})	*Audit trail* (AM, {any}) (SL, { })
System management and audit (AM, {all}) (SL, { })	R	R	R	R	R	R	R	RW
Production users (SL, {P}) (SL, {P})	RW	R				R		W
Application programmers (SL, {D}) (SL, {D})			RW		R	R		W
System programmers (SL, {SD}) (SL, {D})				RW	R	R		W
System control (SL, {P,D}) (SP, {P,D})	RW	RW	RW	RW	RW	RW	RW	W
Repair (SL, {P}) (SL, {P})	RW	R				R	R	W

[a]Category labels are interpreted thus: P = Production; D = Development; SD = System Development; SP = System Program; SL = System Low; AM = Audit-Manager; O = Operational.

The effect of this approach is to enforce a course integrity policy that is supportive of data integrity requirements by preventing access to production data except by production programs whose correctness has presumably been assured. Although it does not tightly bind individual programs to individual data objects, the distinction between production and non-production objects is an important one for many commercial organisations.

2.5 The Trusted Network Interpretation of the TCSEC

The Network Interpretation (TNI) of the TCSEC intends to provide an interpretation of the TCSEC that applies to networked systems [6]. It identifies additional considerations that apply to networks, which are not explictly addressed in the original TCSEC.

The property of integrity in a system receives much more attention in the TNI. The TCSEC considers integrity primarily in terms of the need to protect mandatory security labels against unauthorized alteration, and what is referred to as "overall system integrity" (presumably the integrity of its TCB). The TNI introduces integrity requirements dealing with the need to ensure that information is reliably passed between components of the network. The issues introduced include correctness of message transmission, authentication of the source and destination of a message, and correctness of the various data fields used to transfer user and protocol data.

The TNI observes that the term "integrity" when used in the context of computer systems has been used to refer to such issues as consistency, accuracy, concurrency, data recovery, modification access control and some concept of credibility or quality of information. The TNI concedes that integrity will often be equally if not more important as security (which we can interpret as "confidentiality" in this context), and considers integrity primarily as a property of a system that provides assurance as to the accuracy, faithfulness, non-corruptibility and credibility of information transmitted between source and destination entities.

In this interpretation, an integrity policy addresses both intentional attempts to modify information (referred to as Message Stream Modification), and the unintentional but largely inevitable threat to transmitted information that occurs through noise in communication systems and equipment failure. An integrity policy thus places great emphasis on the ability to write to objects, and constrains modification of information to comply with the integrity policy.

Support for the integrity policy must be provided by a Network Trusted Computing Base (an NTCB), which must ensure that the mechanisms supporting the integrity policy are protected and always invoked. Assurance requirements state that the communication interfaces and protocols used must be specifically tested during evaluation.

Support for an integrity policy in a network system is largely dependent on communication protocols, which must adequately provide for the following.

- Authentication of the entities involved in an interaction, including the point at which components are bound into the system.
- Resilience to component failure.
- Preserving the integrity of a message stream against threats of alteration, deletion, reordering or replay of message sequences. This requires the ability to test, detect and report message stream modifications.
- Preserving message field integrity against similar potential threats.
- Avoidance of deadlock or livelock conditions, including detection of a component being unable to respond correctly (*e.g.* message timeout).

The TNI recognises that the mechanisms used in support of integrity policy requirements are often

probabalistic in nature, and states that the desired probability with which message stream modification can be detected should be specified in the integrity policy. This applies also to the preservation of mandatory security labels under the integrity policy.

The TNI interpretation of mandatory access controls extends the original TCSEC specification by specifically making provision for both secrecy and integrity labels. There are also quite clear criteria laid down for the nature of integrity controls, in that it refers to a single dominance relation supporting mandatory policies, which it suggests are formed by combining secrecy and integrity lattices. This implies that a mandatory integrity policy is expected to be of a similar form to MLS, *i.e.* based on a lattice structure.

The TNI interpretation of integrity is similar to that of the TCSEC in that it addresses integrity of the NTCB, but it also introduces issues of data integrity for data transmitted between NTCB partitions in terms that are similar to those of OSI.

2.6 The Clark–Wilson Model of Integrity

Clark and Wilson [4] published a paper in 1987 in which they argued that the TCSEC did not adequately address the security issues predominant in commercial environments. They argued that these are primarily concerned with preventing unauthorised modification of data as a countermeasure to risks of error and fraud, rather than with issues of information confidentiality. They also argued that countermeasures to these threats have long been established as part of common business practice, principally in the concepts of well-formed transactions and separation of duties, neither of which is directly supported by the TCSEC. They suggested that separate policies are required for confidentiality and data integrity, and that, with the exception of common requirements such as user authentication, much of the mechanism for supporting these two policies would also be different.

The concept of a well-formed transaction is that users cannot manipulate data in an arbitrary fashion, but only in constrained ways which preserve the integrity of the data. This they interpret to a restriction that allows data items to be modified only by a set of programs which have been "certified" to preserve data integrity.

The issue of separation of duty was perceived to improve the correspondence between the system and the real world, on the basis that experience had demonstrated that fraud (*i.e.* a lack of real-world correspondence) is far less likely when collusion between users is required. They interpret this as a requirement to constrain users to a specific set of programs, in such a way that the processing of a complete real-world transaction requires action by more than one user.

By insisting that only security administrators can modify the set of data items which a given program can manipulate, and similarly restricting the ability to define which programs a given user can invoke, it was argued that the policy is essentially mandatory in nature, as opposed to the discretionary nature assumed for such mechanisms in the TCSEC. They recognised that appropriate separation of duty can only be determined with reference to the application concerned, and that this results in a dependence on the interpretation (and trustworthiness) of the security administrator.

The Clark-Wilson paper presented an informal model of a computer system intended to meet the requirements they had defined. Notable entities used in the model are:

- users of the system;
- constrained data items (CDIs), which can only be manipulated by certified programs (TPs);
- transformations procedures (TPs), which are the programs certified to manipulate CDIs in such a way that their integrity is preserved;

- integrity verification procedures (IVPs), which confirm the integrity of all CDIs in the entire system at the time they are run;

- unconstrained data items (UDIs), whose integrity is not assured in the model.

There are two aspects to the model, which are expressed in a number of "certification" and "enforcement" rules applied to the entities above. The certification rules are application specific, and address requirements of real-world correspondence and correctness. They are as follows.

C1. IVPs must confirm the validity of state for the total set of CDIs.

C2. TPs must be certified to take a set of CDIs from one valid state to another valid state (assuming that the initial state is valid).

C3. A relation over the sets of users and TPs must be defined, recognising the principle of separation of duty, identify whether a user can invoke a TP and if so, which CDIs the TP may manipulate on behalf of that user.

C4. TPs must record details of their activity in a log (an instance of a CDI), sufficient for any operation to be reconstructed.

C5. TPs that transform UDIs into CDIs (*e.g.* after validating input data) must be certified to perform only valid transformations.

The certification procedure for IVPs and TPs can only be performed with reference to an integrity policy for the application in question, which presumably expresses correctness requirements in the design of the system.

Enforcement rules also apply to the behaviour required by the system, and are largely independent of a specific application. They are as follows.

E1. The system must enforce the TP↔{CDI} relation defined in rule C2.

E2. The system must store and enforce the relation defined in rule C3.

E3. The system must authenticate user identity.

E4. Only a certified agent can change the relations involving TP entities, and this agent must be prevented from executing such TPs directly.

It was argued that the CDIs will be in a valid state if, after initial confirmation of their integrity by an IVP, only TPs are allowed to manipulate them.

Some of the approaches proposed in solution to this problem are application specific at the grain of various levels in a hierarchy of nested transactions [1], or are mechanisms involving analysis of historic information from an audit trail [15].

The Clark–Wilson model has provoked constructive reaction from within the computer science fraternity and academia. A number of papers have been published that discuss or propose solutions to its requirements, and the United States Institute of Science and Technology (NIST, formerly the National Bureau of Standards) has held invitational workshops to encourage the development and discussion of its basic ideas.

As a result of the deliberations that have taken place since their original paper was published, Clark and Wilson have refined their view of the concepts and mechanisms involved, but still stand by the principles of their original model [13].

Their view of data integrity has been refined to recognise that it involves internal consistency of a system (a correctness aspect) and external consistency (correspondence with reality). They are calling for the development or improvement of a number of control mechanisms to support these requirements.

(1) Operating system support for a triple used to bind users, TPs and CDIs. They argue that contemporary mechanisms support (user, TP) and (TP, CDI) pairs, and that this is inadequate since it makes administration difficult.

(2) Better user authentication mechanisms to replace passwords schemes.

(3) Support for separation of duties through dynamic control, access maps and control of privileged users.

(4) Better techniques for program development and control.

(5) Improved audit trail logging.

(6) Audit trail logs of data changes, with integrity attributes recorded.

(7) Operating systems that enforce automatic segregation of duties and that use message authentication codes to track data changes.

Many of these issues are application specific, or are appropriately addressed by techniques that are not entirely within the domain of operating system issues. This is clearly the case with program development techniques, where software engineering practices are available but are not normally used in commercial data processing environments. It is also largely the case with typing, which a number of commentators have observed is the issue underlying the TP–CDI binding.

The emphasis in Clark–Wilson is distinctly on data integrity, yet is quick to look towards an underlying operating system to provide the primitive mechanisms to support concepts such as the (user, program, CDI) triple, user authentication and separation of duty.

2.7 The Open System View

The International Standards Organisation Open Systems Interconnection standards consider data integrity as a property of data which have not been subject to unauthorised alteration or destruction [14]. These are primarily concerned with communication issues, but many of these will be implemented in operating systems or are fundamental to operating system services in a networked environment.

OSI identifies five security services that may be applied to ensure data integrity:

(1) connection integrity with recovery;

(2) connection integrity without recovery;

(3) selective field connection integrity;

(4) connectionless integrity;

(5) selective field connectionless integrity.

A fundamental mechanism used in providing these services is to include "black box codes" or "cryptographic checkvalues" in messages sent over a communication channel, which the receiver can recalculate to determine whether the message has been modified since it was sent. These techniques are fundamental in assuring data integrity during transfers over telecommunication channels, but additional mechanisms may need to be implemented in communication protocols to detect the deletion, replay, insertion or reordering of messages (individually or as part of a sequence).

2.8 Partially Trusted Subjects

Lee [16] has described a solution to the Clark–Wilson requirements based on the use of the Biba strict integrity lattice, using labels consisting of integrity categories only (no level components). His solution controls modification of data, and uses "partially trusted" subjects to implement the Transformation Procedures of Clark and Wilson.

In this scheme, each subject has two integrity labels associated with it. The "view-minimum" label for a

subject, v-min(s), specifies a minimum set of integrity categories required in the label of objects that s wishes to read. The "alter-maximum" label, a-max(s), must dominate (be a superset of) the integrity categories in the label of any object s wishes to write. Untrusted subjects have v-min(s) = a-max(s) as in the original Biba model (equivalent to having only a single integrity label). Partially trusted subjects are still constrained by the integrity policy, but can read any object whose integrity label dominates v-min(s) or a-max(s) (the latter allows the object to read any object it can write, which is also less restrictive than the strict Biba policy allows). This represents a relaxation of the strict integrity policy, since a subject can be permitted to read objects of lower integrity than those to which it writes. Its ability to perform this is limited by the v-min(s) and a-max(s) labels, hence the term "partially trusted subject".

Lee describes a method for applying an integrity lattice with partially trusted subjects to the solution of the Clark-Wilson problem.

(1) Integrity categories are associated with each type of operational user.

(2) Integrity categories are associated with each type of CDI.

(3) Integrity categories are associated with each type of administrative and development role. The three sets of integrity categories are disjoint, which prevents, for example, a user creating untrusted code that can manipulate production data.

(4) Conflicting roles are identified, so that dangerous sets of integrity categories can be avoided.

(5) Production code is given a "production software" category, which no user can log on with (similar to Lipner's approach).

(6) Certified TPs are labelled as partially trusted subjects, where v-min(s) is the role (label) of users who can invoke the set of CDI types it can modify.

(7) Security authorising transactions are not permitted to operational users under rule (4).

(8) The system can be used to ensure that software cannot masquerade as an IVP.

In addition to considering the Clark-Wilson problem, Lee made a number of observations regarding potential complications for integrity policies.

(1) It may be useful to have a limited form of hierarchic category, to simplify situations such as making system code available to all types of process (*e.g.* system > production > development).

(2) It may be useful to represent the v-min label as a set of sets, any one of which may be dominated by an object being read. This would remove the need to allow reading of objects that dominate a-max(s).

A potential problem with the use of labels in such a scheme is the large number of TPs that may exist compared with the limited number of categories available [26]. It has been suggested that this is particularly difficult since categories tend to be managed centrally, while many commercial policies require decentralized management.

The scheme attempts to narrow the gap between the data integrity objectives of Clark and Wilson and the available controls provided through Biba mechanisms. The additional flexibility is obtained at the expense of weakening the Biba strict integrity policy, but is still a relatively coarse form of control.

2.9 WIPCIS II

The invitational Workshop on Integrity Policy in Commercial Information Systems run by the National Institute of Science and Technology included in operating systems workshop to discuss integrity implications and mechanisms in operating systems.

The workshop observed that integrity controls could extend well beyond an operating system TCB and that many of these controls could be completely invisible to an operating system, *i.e.* they are application issues. Similarly, integrity threats were perceived as being largely application specific, and those that apply directly to operating systems are covered by the TCB self-protection requirements of the TCSEC.

They concentrated on identifying integrity control objectives and mechanisms that could be provided by operating systems. Some of these were deemed necessary, in that it is clearly the operating system's responsibility to provide them, and others were deemed optional, in that it was considered perfectly reasonable for provide them from some other area.

The necessary operating system control objectives were to

- protect objects from unauthorised modification;
- protect programs from unauthorised execution;
- bind data to programs;
- support accountability;
- support improved user authentication mechanisms;
- support application level audit trails;
- support separation of duty between privileged users such as system administrators, security officers, operators and auditors;
- support assurance requirements;
- facilitate the correct and safe use of a system.

The latter point was considered to be important because of the perception that many contemporary systems have significant security exposures resulting from incorrect installation, use or maintenance of the system.

Optional control objectives for operating systems were identified as

- binding integrity information to objects;
- support of strict sequencing between application programs;
- support for dynamic separation of duties.

In considering the mechanisms that operating systems might provide to meet these objectives, it was observed that the Biba approach is supported by several existing systems and provides a "natural" mechanism for controlling object modification at the level of abstraction seen at the operating system boundary. There was concern that the Biba model was not intended to support partitioning of a change or separation of duty, and that the problem with Lee's extensions was the need for trusted subjects. It was concluded that mechanisms supporting strong typing or assured pipelines offer the most promising solutions for the future.

Necessary and optional features of operating systems were identified to meet the control requirements above. The necessary features are

- mandatory integrity controls (in the sense that users have no choice in the matter);
- binding of users to programs, and programs to data (the user of (user,program) and (program,data) pairs was believed to offer better granularity than the ⟨user,program,data⟩ triple promulgated by Clark and Wilson);
- interfaces to support advanced user authentication devices;
- interfaces supporting application audit trails;
- structuring of privileged user functions to support separation of duty.

Optional features were identified as

- integrity tags, binding integrity information to objects;
- support for sequencing of operations;
- support for dynamic separation of duties;
- enhanced user and manager interfaces offering better default values, clearer structure and user friendliness.

These findings are interesting in that they present a consensus view on how far operating systems should go in providing mechanisms to support data integrity requirements. There is an apparent desire to provide only simple primitives from which more elaborate schemes can be constructed.

2.10 A View on Software Development

Sennett makes some interesting observations concerning integrity [22] (note that the use of the term "integrity" in the title of the referenced work is used to cover a wide range of issues relating to high quality software engineering, which is the main theme of the work). These concern the use of integrity mechanisms to control modification of software during the development life-cycle, and the existence of a hierarchy of control mechanisms.

The integrity mechanisms described for the control of software development uses the conventional subset lattice, requiring that a user's integrity clearance dominate (be a superset of) an object's integrity label if the user wishes to write to the object. If it is required to restrict update operations to certain tools, then the greatest lower bound of the user's label and that of the tool (*i.e.* the intersection of their integrity labels) must dominate that of the object. Clearly, if a tool is not labelled with the necessary integrity categories, it cannot be used to manipulate an object.

Sennett argues that further control is not necessary, and in particular that the integrity *-property is unnecessary and responsible for practical difficulties in the use of integrity controls. He argues that the integrity of data results from the process it has undergone, *e.g.* compilation, and that integrity controls preventing a compiler from writing at a higher level than that of the data it has read (*e.g.* data without an "executable" integrity category) would be inappropriate. He states that avoiding the integrity *-property avoids the problem of data tending to end up with only high or low integrity (as a result of over- or under-classification). He also states that an integrity policy is not required to correspond to the real world, so his interpretation may appear to be degenerate compared with some of the other views prevalent at present.

Sennett observes that mandatory integrity controls can be supplemented by a typing scheme, and that in general the controls required in a software development environment can be arranged in the following hierarchy offering increasing levels of control:

(1) discrepancy checking (error detection codes and similar techniques);

(2) discretionary access control;

(3) mandatory integrity control;

(4) type checking.

The intention is apparently to assure the data integrity of program code, and a scheme based on mandatory labelling is described for achieving this. It is presented as a more reliable mechanism than discretionary access control, but identifies that strong typing mechanisms are better still.

2.11 A "New" View of Integrity

Terry and Wiseman [23] have presented a "new" model of security in a paper that provides some illuminating observations on the nature of security policies in general and the Clark–Wilson notion of the data integrity in particular. They suggest that notions of security have tended to emphasise one

aspect of security, typically confidentiality or data integrity, purely because of the limitations of current technology. They suggest that there is no natural distinction between these various aspects, and that arguments based around such a distinction may be artificial (in particular, comparisons between "military" and "commercial" requirements). They argue that there are two different notions involved in the Clark–Wilson "model" of data integrity.

The first notion, for which they reserve the term "integrity", is concerned with the internal correctness of a system, and can be used to prevent users from causing effects for which they are not authorised. This is the "error control" objective of the Clark–Wilson model, and is supported by their TP mechanism (which Terry and Wiseman suggest is an issue of typing).

The second notion is concerned with correspondence to the real world, which they call "appropriateness". This is concerned with whether an action is a desirable or "right" thing to do, irrespective of whether it is authorised, and addresses the Clark–Wilson objective of fraud control. They do recognise that correspondence with the real world is difficult to model or to enforce, but accept that separation of duties is an effective mechanism in practice.

They observe that the Biba model supports a coarse notion of typing in its integrity categories, but are critical of its hierarchic integrity levels which they describe as a solution requiring a problem. This observation is interesting because it supports the ordering of labelling and strong typing in Sennett's hierarchy of controls.

3. A Correlated View

This section offers an interpretation of integrity issues based on the observations in the preceding section. It seeks to identify common themes that appear in different authors' views, since these are likely to be the genuinely fundamental issues.

3.1 The Notion of Integrity

The view that integrity is concerned with preventing unauthorised modification of information recurs frequently in the work reviewed above, and is the interpretation adopted in the remainder of this paper. It is clear that the issue is easily confused with other desirable properties of an operating system, which we will try to avoid by using other terms for these properties. Much of the confusion that can arise does so precisely because of failure to distinguish between the different properties desired of a system.

Firstly, the distinction between data integrity at an application level and notions of integrity that occur at more primitive levels within a system is important. The distinction is often related to the level of abstraction at which we choose to consider a system. This determines the grain of the objects and operations that are visible, and influences the interpretations we are able to apply to data.

It is useful to distinguish between "integrity" and "correctness", which is frequently confused when speaking of a system which correctly implements its specification. Whenever a high degree of assurance is required for a system required to enforce an integrity policy, issues of correctness arise, for example in the "certification" rules of Clark and Wilson (although their treatment of the issue is relatively superficial).

Next, it is useful to distinguish between integrity and "appropriateness", in the sense defined by Terry and Wiseman. This helps avoid confusion between real-world correspondence, which is often related to the functions provided at an application level (for example the Clark–Wilson "separation of duty" requirement), and the abstractions (and therefore policy) which we can expect an operating system to implement.

We will use the term "data integrity" to describe a property of data for which we have some assurance that it has not been modified in some unauthorised or undesirable manner. This is useful when dis-

cussing passive protection mechanisms that allow us to detect that a data item has been modified during storage or during transmission.

Using this interpretation, the remainder of this section considers the requirement and enforcement of integrity in operating systems.

3.2 TCB Integrity

The concept of TCB integrity appears in Biba's "ring" policy and in IBM's "statements of integrity", and it is fundamental at all levels of the TCSEC. It is concerned with protection of the content and boundary of the TCB against unauthorised modification, *i.e.* with the integrity of the TCB as a typed object.

TCB (or NTCB) integrity appears to be fundamental to other types of integrity that may be required, evidenced for example by those viruses that freely modify operating systems for which TCB integrity is not assured. The principal reason for this is that the TCB is often used as an enforcement mechanism for integrity policies that are applied beyond the TCB boundary.

Increasing levels of assurance are provided by the TCSEC evaluation levels, and there is no obvious difference between "military" and "commercial" requirements in this area. The correctness of TCB implementation is critical in the preservation of TCB integrity, since flawed controls at the TCB boundary (notably incomplete or fragile parameter validation) can allow all forms of software based control to be undermined [8, 17]. A major concern in this area has to be local modifications to the TCB [19, 20], and TCB software maintenance levels that have not been evaluated.

3.3 Integrity Policy

An integrity policy describes requirements that govern the modification of data beyond the TCB boundary. It is useful to limit the domain of the policy to those objects that are properly part of the system, so as to avoid confusion with issues of appropriateness. It also appears genuinely useful to distinguish TCB integrity from the more general requirements of an integrity policy, although it can be argued that a general integrity policy implictly covers TCB integrity.

When the objects containing data and the subjects manipulating it are created and managed by the operating system (*e.g.* files, memory segments and processes), it is appropriate for the operating system to provide mechanisms for policy enforcement. When the entities constrained by the integrity policy are more abstract than those managed by the operating system, then the operating system can often do no more than provide support for abstract mechanisms which are responsible for policy enforcement. Operating system enforcement of integrity policy is therefore likely to be limited in terms of level of abstraction and policy grain.

For example, the Clark–Wilson view involves issues of typing and appropriateness. It is also significantly more abstract than most operating systems could directly enforce, and is hence often seen as an application issue. The concern then is to identify how the operating system can provide support for the application enforcing the finely grained integrity policy, *e.g.* using Biba's labelling scheme to enforce coarsely grained typing. The danger is in not recognizing the different levels of abstraction and the different policies involved, which appears to be the case with the Clark–Wilson ⟨user,program,CDI⟩ triples. The (program,CDI) binding is a form of typing, and is largely an issue for the system designer. The (user,program) binding is essentially an issue of appropriateness, and is largely an issue for the application administrator. If we choose to enforce these distinct requirements with a mechanism that confuses this distinction, the result is likely to be a less flexible mechanism (the WIP-CIS view) and one that is more difficult for both parties to use effectively.

In contrast, the Biba strict policy provides coarse type enforcement and can remove some of the responsibility for integrity policy enforcement from the application. If the integrity *-property is

relaxed, *e.g.* as suggested by Sennett, the result is to increase application responsibility for policy enforcement, limiting assurance of overall policy enforcement to those aspects of the system's behaviour that the application can detect and control.

There is no obvious distinction between "military" and "commercial" integrity policies. In both cases they describe a relationship between the information flows in a computer system and, at the level of abstraction of a TCB boundary, the entities and operations concerned are the same in both cases.

There is, however, a need for caution when discussing "commercial" policies. Lipner and Clark and Wilson both describe "commercial" requirements, but the policies they describe are different. There is no consensus from the commercial sector that either policy is adequate. Indeed, both appear to have originated with financial auditing circles, and do not necessarily express the views of developers or users (indeed they do not necessarily express the views of the financial auditing community at large).

3.4 Passive Data Integrity Mechanisms

The data integrity mechanisms are at the bottom of Sennett's hierarchy of controls, and are essentially passive in nature (they allow us to detect, but not to prevent, data modification). They occur in the error detection codes and protocols of the TNI and OSI, as well as in the Message Authentication Codes of the TNI, OSI and Clark and Wilson.

In most cases the mechanisms depend upon the use of an error detection code whose length is considerably less than that of the message it is associated with (typical lengths being 16, 32 or 64 bits). These techniques are therefore probabalistic, since a strong code will detect message modification with a probability no greater than $1-2^{-n}$ where n is the size of the code in bits. An integrity policy may therefore need to specify the probability with which data modification is to be detected.

The matter is further complicated when we consider the nature of threats to data integrity if we wish to determine how reliable a code is as a protection mechanism. In the case of random errors to data, there may be nothing to choose between strong "block check codes" and "cryptographic checkvalues" of the same length. If deliberate modification of messages is a threat, then the use of "block check codes" often provides little assurance since a new code may be easily calculated following a message modification, or a message may be modified in a way calculated to produce the original checksum. In such cases a "cryptographic checksum" may be used, which by definition makes it difficult for these attacks to succeed. Thus the choice of integrity mechanism is dependent upon the nature of threats to data integrity.

In terms of policy and specification, it is useful to regard the transmission or storage of data as a process which reduces data integrity, and the operations that validate transmitted data (using error detection, error correction and supporting protocols) as processes which raise data integrity.

3.5 Mandatory Integrity Controls

The use of mandatory access control based on integrity labels, as described by Biba and Lee, appears to offer a promising mechanism for integrity policy enforcement at the TCB boundary. Lee's extensions offer a solution to the problems identified by Sennett (a compiler needs to write executable objects, having read objects that are not executable) and Biba (relaxing the constraint on the availability of data that can be read). Lipner's observation that his "repair" programs do not write data with the integrity required for production applications is precisely the sort of control desired, since the function of the repair code has not been "certified" to preserve the integrity of the objects written (in the Clark–Wilson sense).

Programs that result in an increase in the integrity of an information stream are extremely common, as discussed below. The practicality of mandatory integrity controls which do not provide for partially trusted subjects or some equivalent concept, is unclear. The issue of how integrity labels should

be assigned to objects and subjects is largely an issue of appropriateness. The results of Lipner and Lee are both encouraging in the sense that they demonstrate that integrity lattices can model different requirements.

The comment (by Terry and Wiseman) regarding the redundancy of hierarchic integrity levels does not lead to weakening of the concept. Indeed, given a non-hierarchic subset lattice we can easily model hierarchic ordering. For example, if we associate a category with each hierarchic level, and mark objects with the category labels associated with their intended level and all the lower levels, then the subset relation between object labels imposes a hierarchic ordering. In this sense the hierarchic component of MLS security labels is equally redundant.

Unfortunately there is little evidence of the practicality and utility of mandatory integrity controls. Both Biba and Lipner indicated that practical experience is needed in this area, but there are no accounts of this available.

4. An Additional Aspect of Integrity

We have mentioned that operations leading to an increase in integrity are widespread. This section expends upon this view.

A large number of programs are required to read data that they do not entirely trust. The first, and sometimes only, operation of most such programs is to "validate" this data, by checking its consistency or comparing it with data obtained from elsewhere. The effect of this is rejection of some data, and increased confidence in the data which passes the tests. When viewed as part of a series of information flows within a system, the data which passes the tests has greater integrity than the data which was entered for testing, and programs which perform this upgrading are "partially trusted" in the sense of Lee's exposition.

When we start to consider the extent with which such operations are performed, it becomes apparent that changes in integrity level of information are common, so it is not suprising that mechanisms for enforcing integrity policy are likely to be overly restrictive if they do not recognise this. For example, in the case of a compiler, both source and object versions of a program describe the same set of operations to be performed on the same set of data. We intuitively regard object versions as having the greater integrity on the basis that the encoded algorithm they represent has at least passed the syntactic checks of a compiler.

This view is consistent with that found in ref. 22 (p. 340), where it is argued that the integrity of data results from the processes it has undergone, rather than from some intrinsic property of the data. In this case we are suggesting that the process which results in an increase in the integrity of data is testing.

To understand the nature of the validation process, it is useful to consider a system in terms of the information flows and repositories it contains. A fundamental feature of the information flows in validation processes is redundancy, which can occur in a variety of guises. Redundant information allows us to perform cross-checks, *e.g.* in the case of error detection codes, and provides some assurance of data correctness. It applies, for example, in the data processing techniques of comparing input record fields against master files and even in techniques that handle information flows from beyond the system boundary, such as data entry verification (where data are literally keyed in twice).

A surprising aspect of redundancy is therefore its usefulness as an appropriateness technique. This is essentially another way of interpreting the Clark–Wilson separation of duty (*i.e.* as a command stream with a high degree of redundancy).

5. Relevance to Contemporary Systems

There is clearly a need for improved integrity controls in most existing operating systems, many of which prove to be unreliable in practice.

The influence of the TCSEC is such that many suppliers are improving their products to conform with higher evaluation classes [12], examples being MVS/ESA with RACF 1.9 [11], VME with the High Security Option and UNIX System V with Enhanced Security [24]. This results in greater attention to TCB integrity, although the assurances this provides can be undermined as a result of software maintenance or functional extensions to the TCB (see above). It also results in the availability of mandatory access controls in systems at and above class B1, although these may be limited to MLS security controls.

Systems such as VME with the High Security Option provide mandatory integrity controls, as described by Biba, in addition to the MLS controls. The TCSEC does not specifically require mandatory integrity controls, and systems such as MVS/ESA and UNIX System V Enhanced Security do not provide them [24]. The influence of the TCSEC encourages the introduction of label-based access controls, while the trend towards object-oriented approaches encourages support for typing mechanisms. Operating systems such as VME and OS400 provide some support for typing, and the security of these systems appears to benefit from this approach in practice.

It is desirable to achieve improvements using whatever tools are available, and the MLS security controls can be used to achieve support for a mandatory integrity policy if labels are chosen correctly. This can be achieved as follows.

- Decide upon the integrity labels required.

- For each object and each subject, select the appropriate set of integrity labels to enforce the required policy.

- Do not mark the objects and subjects with the selected set of labels, instead mark them with the complement of that set from the complete set of integrity labels.

The effect of using the complement of the selected integrity labels is that the MLS subset operator will enforce the desired integrity relation over the set of labelled objects and subjects. The scheme is limited in that it will prove considerably more difficult to manage, particularly in the event that the complete set of integrity labels needs to be extended. This limitation could largely be overcome by the provision of tools for its users, and is important in those cases where the required access control mechanism is not otherwise available. UNIX System V Enhanced Security uses this approach to enforce an integrity policy for its TCB, by setting all TCB code to system low, with all other processes running above this level. The effect is that TCB code is made available under the mandatory controls to all processes for read access, but also ensures that these processes cannot alter the TCB code.

The remaining obstacles to be overcome concerning the practical use of label-based integrity controls are the lack of familiarity that most security administrators have with the subject matter and the lack of documented experiences in attempting to deploy mandatory integrity controls to achieve commercial requirements. There are indications that such controls are awkward to configure correctly and that they can be over-restrictive, so that the administrative burden incurred through their use should not be underestimated.

Integrity controls based on the Biba approach provide a coarse form of control, whereas type-based approaches offer a much more refined approach. Typing schemes limit the actions that can be performed on a data object of a given type to a specific set of operations. These operations are typically declared at the time the data type itself is defined, and it is sometimes appropriate to consider a data type entirely in terms of the operations defined on it. The operations typically include constructor and destructor functions for creating and destroying instances of the data type. In principle, the data object may only be accessed using the operations defined for the data type, so that the way the data

are actually stored may be hidden from all other programs.

Strong typing is inherent in a variety of recent concepts, including abstract data types, mathematical data types, classes and object-oriented approaches. Many of these are supported by language systems, and run-time enforcement of typing can be provided by mechanisms such as monitors, protected subsystems and object managers. Computer architectures that allow many small protection domains to be implemented provide a good environment for implementing these mechanisms, and in particular capability-based protection mechanisms have been successfully used to this end.

The intentions of strong typing and the Clark-Wilson approach are similar in that they aim to preserve properties of data by constraining the ways in which it can be manipulated. Clark and Wilson are concerned with typing at a high level of abstraction, and whether an operating system can enforce this depends upon the design of the application and how this design maps onto objects at level of the operating system interface. If several Clark-Wilson CDIs are implemented in a single operating system file, *e.g.* in a shared database, then the operating system is unlikely to support directly the typing of the CDIs.

6. Conclusion

Much of the confusion relating to issues of integrity can be avoided by being more precise about the properties of systems under discussion. There are a number of different interpretations given to the term but the subject matter is often closely related. For example, Clark and Wilson discuss issues of data integrity at an application level while introducing issues of operating system integrity and data integrity in the OSI sense (in respect of message authentication codes). The term "integrity" fits all of these notions naturally, but frequently becomes heavily overloaded. The precise intent of the term needs to be clearly established in the context of the system under discussion and the precise properties desired of that system.

It appears that the use of mandatory integrity controls provides some potential for improved support for integrity policies. The deployment of such controls has been proposed in the literature, and these proposals have not yet been undermined. Moreover, they appear to be consistent with the controls we can expect to appear in operating systems, and the basic mechanics are being provided in many contemporary operating systems.

It also appears that the TCSEC is a positive and desirable influence, resulting in improvements to TCB integrity and has encouraged the availability of mandatory integrity controls (although not as directly as it might have done).

The most extraordinary aspect of mandatory integrity controls based on labelling is the lack of available experience in deploying them. The basic mechanism was described over a decade ago, but has not resulted in significant interest in attempting to use such controls effectively or to discover and relate its strengths and limitations. This is possibly due to a lack of research funding for "integrity" issues in comparison with the support and interest in confidentiality aspects of security promoted by military bodies.

From a pragmatic viewpoint, the increasing availability of mandatory integrity controls presents an obvious opportunity, and exploring how to deploy these mechanisms effectively appears to be a worthwhile challenge.

References

[1] L. Badger, A model for specifying multi-granularity integrity policies, *Proc. 1989 Symp. on Security and Privacy*, IEEE, New York, 1989, pp. 269–277.

[2] D. E. Bell and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, MITRE, Bedford, MA, 1977.

[3] K. J. Biba, *Integrity Considerations for Secure Computer Systems*, USAF Electronic Systems Division, Bedford, MA, 1977 (ESD-TR-76-372).

[4] D. D. Clark and D. R. Wilson, A comparison of commercial and military computer security policies, *Proc. 1987 IEEE Symp. on Security and Privacy*, IEEE, New York, 1987, pp. 184–194.
[5] *Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense Computer Security Center, Fort George G. Meade, MD, 1983 (CSC-STD-001-83).
[6] *Trusted Network Interpretation of the TCSEC*, The National Computer Security Center, Fort George G. Meade, MD, 1985 (DoD 5200.28-STD).
[7] *Security Functionality Manual*, DTI Commercial Computer Security Centre, V21-Version 3.0 (unpublished).
[8] B. Hebbard *et al.*, A penetration analysis of the Michigan terminal system, *ACM Operating System Rev.*, 14 (1) (1980) 7–20.
[9] *Statement of MVS System Integrity*, International Business Machines, October 1981 (IBM ULET ZP81-0801).
[10] *Statement of System Integrity for VM/System Product Release 3 and Companion VM/SP High Performance Option Release*, International Business Machines, IBM Programming Information Customer Letter, June 1983 (IBM ULET 5664-147, 5664-173).
[11] *Multi-Level Security Statement of Direction*, International Business Machines, March 1988 (IBM ULET ZA88-0147).
[12] *System Programming Library: System Macros and Facilities*, Vol. 1, International Business Machines, pp. 193–204.
[13] D. R. Wilson, The integrity initiative—where are the auditors? *COMPACS '90, Proc. 14th Int. Conf. on Computer Audit, Control and Security*, The Institute of Internal Auditors, U.K., March 1990.
[14] *OSI Reference Model—Part 2: Security Architecture* (DIS 7498-2; ISO/TEC JTC 1/SC21N).
[15] P. A. Karger, Implementing commercial data integrity with secure capabilities, *Proc. 1988 Symp. on Security and Privacy*, IEEE, New York, 1988, pp. 130–139.
[16] T. M. P. Lee, Using mandatory integrity to enforce "commercial" security, *Proc. 1988 Symp. on Security and Privacy*, IEEE, New York, 1988, pp. 140–146.
[17] R. R. Linde, Operating systems penetration, *National Computer Conf., 1975*, AFIPS Press, Montvale, NJ, 1975, pp. 361–366.
[18] S. B. Lipner, Non-discretionary controls for commercial applications, *Proc. 1982 Symp. on Security and Privacy*, IEEE, New York, 1988, pp. 2–10.
[19] R. Paans and G. Bonnes, Surreptitious security violation in the MVS operating system, *Comput. Secur.*, 2 (1983) 144–152.
[20] R. Paans and I. S. Herschberg, How to control MVS user supervisor calls, *Comput. Secur.*, 5 (1986) 46–54.
[21] M. D. Schroeder, D. D. Clark and J. H. Saltzer, The Multics kernel design project, *ACM Operating Systems Rev.*, 11 (5) (1977) 43–56.
[22] C. Sennett (ed.), *High-integrity Software*, Pitman, London, 1989.
[23] P. Terry and S. Wiseman, A "new" security policy model, *Proc. 1989 IEEE Symp. on Security and Privacy*, IEEE, New York, 1989, pp. 215–228.
[24] *UNIX System V Security Today and Tomorrow*, UNIX International, NJ, 1990.
[25] F. Cohen, Computer Viruses, *Comput. Secur.*, 6 (1987) 22–35.
[26] P. A. Karger, Implementing commercial data integrity with secure capabilities, *Proc. IEEE Symp. on Security and Privacy, 1988*, IEEE, New York, 1988, pp. 130–139.

Greg O'Shea is a consultant in the IT Security Group at KPMG Peat Marwick McLintock. Before this he was Systems Programming Manager at a leading British merchant bank, working with mainframe computers from a variety of suppliers. In his spare time he is a research student at Birkbeck College, University of London. He is the current Secretary of the Computer Security Specialist Group of the British Computer Society.