FOREWORD

This report is the fourth of five companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses auditing issues in secure database management systems.

[illegible]

May 1996

Keith F. Brewster
Acting Chief, Partnerships and Processes

ACKNOWLEDGMENTS

The National Computer Security Center extends special recognition to the authors of this document. The initial version was written by Richard Graubart of the MITRE Corporation. The final version was written by Stan Wisseman, Bill Wilson, and David Wichers of Arca Systems, Inc.

The documents in this series were produced under the guidance of Shawn P. O'Brien of the National Security Agency, LouAnna Notargiacomo and Barbara Blaustein of the MITRE Corporation, and David Wichers of Arca Systems, Inc.

We wish to thank the members of the information security community who enthusiastically gave of their time and technical expertise in reviewing these documents and in providing valuable comments and suggestions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

This document is the fourth volume in the series of companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria* [TDI 91; DoD 85]. This document examines auditing issues in secure database management systems and summarizes the research to date in this area.

1.1 BACKGROUND AND PURPOSE

In 1991 the National Computer Security Center published the *Trusted Database Management System Interpretation* (TDI) of the *Trusted Computer System Evaluation Criteria* (TCSEC). The TDI, however, does not address many topics that are important to the design and development of secure database management systems (DBMSs). These topics (such as inference, aggregation, and database integrity) are being addressed by ongoing research and development. Since specific techniques in these topic areas had not yet gained broad acceptance, the topics were considered inappropriate for inclusion in the TDI.

The TDI is being supplemented by a series of companion documents to address these issues specific to secure DBMSs. Each companion document focuses on one topic by describing the problem, discussing the issues, and summarizing the research that has been done to date. The intent of the series is to make it clear to DBMS vendors, system designers, evaluators, and researchers what the issues are, the current approaches, their pros and cons, how they relate to a TCSEC/TDI evaluation, and what specific areas require additional research. Although some guidance may be presented, nothing contained within these documents should be interpreted as criteria.

These documents assume the reader understands basic DBMS concepts and relational database terminology. A security background sufficient to use the TDI and TCSEC is also assumed; however, fundamentals are discussed wherever a common understanding is important to the discussion.

1.2 SCOPE

This document addresses audit in secure DBMSs. It is the fourth of five volumes in the series of TDI companion documents, which includes the following documents:

- *Inference and Aggregation Issues in Secure Database Management Systems* [Inference 96]
- *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96]
- *Polyinstantiation Issues in Multilevel Secure Database Management Systems* [Poly 96]
- *Auditing Issues in Secure Database Management Systems*
- *Discretionary Access Control Issues in High Assurance Secure Database Management System*s [DAC 96]

This series of documents uses terminology from the relational model to provide a common basis for understanding the concepts presented. This does not mean that these concepts do not apply to other database models and modeling paradigms.

1.3 INTRODUCTION TO AUDITING

Auditing has long been recognized as being a critical element in secure systems. Schaefer et al., examined the historical significance of auditing in [Schaefer 89b]. That discussion is reused here.

> The word ‘‘audit” is derived from the Latin *auditus* (hearing) and is defined by the *Oxford English Dictionary* as an ‘‘official examination of accounts with verification by reference to witnesses and vouchers. To make an official systematic examination of (accounts) so as to ascertain their accuracy.” From the earliest citations until the present epoch there is a close relationship between the concepts of *audit, accountability, accounting,* and *accuracy.*
>
> The original purposes of audit focused on a quest for establishing the verity of account books and ledgers. A primary interest of the auditor was to ascertain whether the reckoning of accounts represented on paper accurately depicted what could be tied to reality and, if not, why not. ...
>
> In the operating system context, the original intention of audit appears to have begun with an investigation into the accuracy and legitimacy of charges to customers for the use of computing resources. The system of ‘‘witnesses and vouchers” checked by Electronic Data Processing (EDP) auditors consisted largely of a system-generated ‘‘accounting log’’ whose entries associated a specific user account with an identified use of some billable resource (e.g., machine store, magnetic tape, CPU time, printer, card punch, etc.) on a specific date and time for some time period....
>
> The accuracy of bills clearly related to the accuracy of the accounting log. If some billable events were not recorded, the computer center would lose revenue. If some events were entered into the accounting log that either did not occur or were caused by another user’s actions, some customer would be overbilled. If some customer were capable of altering the accounting log entries, some form of fraud would occur.
>
> The TCSEC audit requirements are derived fairly directly from the needs of EDP auditors. A set of accountable events is defined, and each such must be associated with an accountable user and accurately recorded into a protected security audit log along with the appropriate ‘‘witnesses and vouchers” to the event’s context.

In an automated information system (AIS), the audit trail is a chronological record of when users log in, how long they are engaged in various activities, what they were engaged in, and whether an actual or attempted security violation occurred [Stang 93]. The record should be sufficient to enable the reconstruction, review, and sequence of environment surrounding or leading to each event in the path of a transaction from its inception to output of final results. Such trails may be examined by a system security officer (SSO). They may also be examined, depending on the circumstances, by an operating system (SO) security specialist, application specialist, or by an information security (INFOSEC) specialist.

1.4 AUDIENCES OF THIS DOCUMENT

This document is targeted at four primary audiences: the security research community, database application developers/system integrators, trusted product vendors, and product evaluators. In general, this document is intended to present a basis for understanding secure DBMS auditing. Members of the specific audiences should expect to get the following from this document:

<u>Researcher</u>

This document describes the basic issues associated with secure DBMS auditing. Important research contributions are discussed as various topics are covered. This discussion will assist the research community in understanding the scope of the auditing issues and highlight areas that require additional research.

Database Application Developer/System Integrator

This document highlights the potential hazards and added management complexity resulting from secure DBMS auditing. This document discusses the ramifications a secure DBMS audit trail has on storage, performance, and analysis tools.

Trusted Product Vendor

This document identifies likely enhancement paths that would add value to existing audit mechanisms, yet remain TCSEC compliant. It provides the basis for understanding how audit requirements have been interpreted for secure DBMSs in previous NCSC evaluations and how various architectures affect auditing.

Evaluator

This document presents an understanding of auditing issues so that evaluators can better evaluate secure DBMS audit mechanisms. It accomplishes this by providing TCSEC/TDI references where applicable, and examples of Trusted DBMSs where available.

1.5 ORGANIZATION OF THIS DOCUMENT

The organization of the remainder of this document is as follows:

- Section 2 provides the definition and purpose of audit, and compares secure DBMS auditing to that of a secure OS.
- Section 3, based on a DBMS audit policy, discusses what actions to audit, objects to audit, and the data to be collected about each event.
- Section 4 addresses issues of what DBMS data to collect for each auditable event, how to capture that data, where to store the data, and how much to store. Section 4 also discusses performance issues and audit trail credibility/protection.
- Section 5 addresses ways to facilitate use of DBMS audit data for misuse detection and damage assessment.
- Section 6 describes the impact of variations in secure DBMS architectures on assurance and other aspects of auditing.
- Section 7 introduces the concept of recordability, the ability to precisely adjust the amount of data that is recorded for particular audit events.
- Section 8 summarizes the contents of this document.

SECTION 2

BACKGROUND

A broad definition of audit is that it is the collection and review of a documented history of the use of a system to verify that the system is intact and working effectively [DoD 84]. This section further examines the definition and purpose of auditing. It then compares secure DBMSs to secure OSs, emphasizing those differences that impact auditing. Finally, some of the audit issues examined in this document are enumerated.

2.1 AUDIT DEFINITION

The auditing activity, viewed in its entirety, encompasses the notion of both an *interna*l and *external* audit.

An *internal audit* is distinguished from an external audit by the fact that internal auditing is designed into the system and runs continuously, whereas an external audit is only performed periodically and by means external to a system. Internal audit consists of automated collection and analysis of documentary evidence of the system's use. An internal audit subsystem must include mechanisms for continuously collecting and recording audit data and for periodically reviewing and analyzing the collected data. The internal audit collection mechanism records data about system activities that have been judged significant to audit; these are referred to as audit events. An audit event may originate in the operating environment (e.g., a network connection to a DBMS), in the operating system (e.g., as a file open), or in the DBMS (e.g., as a database query). The same thread of activity may be audited at three different levels of abstraction with correspondingly different granularity. Data collected about the occurrence of audit events is either recorded in the respective audit trails, or merged together into a composite audit trail. Multiple audit trails may need to be compared and reconciled during the investigation of a suspected or actual security breach. Since it serves as the only internal accountability mechanism, the audit trail, the audit settings, and the audit mechanism itself must be trusted/protected—comparable to the human external auditor's auditing instructions and audit results.

An *external audit* consists of gathering and analyzing documentary evidence about the system by methods that are external to the system. External audit has two facets: (1) operational audit and (2) systems audit. Operational audit is a review of computer operations that covers system security policy, data integrity controls, system development procedures, and backup recovery procedures. Systems audit is an indepth audit of a particular system for accuracy. By tracing the progress of special transactions as they flow through the system auditors gain evidence to certify the accuracy and accountability of the system. The audit process and the collection of documentary audit evidence provide assurance that the system under inspection is functioning properly.

This document discusses internal auditing rather than external, particularly the internal auditing collection mechanism, the automated aspects of the internal audit process, and the audit trail of documentary evidence, all with regard to secure DBMSs.

2.2 PURPOSE OF AUDIT

The *Guide to Understanding Audit in Trusted Systems* points out that the purpose of audit is the same regardless of the computer component performing the audit [Audit 88]. That is, the audit objective does not change just because the component performing the audit is a DBMS rather than an operating system. The TCSEC control objective for accountability is:

> Systems that are used to process or handle classified or other sensitive information must assure individual accountability whenever either a mandatory or discretionary security policy is invoked. Furthermore, to assure accountability the capability must exist for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time, and without undue difficulty.

The *Guide to Understanding Audit in Trusted Systems* specifies five auditing goals:

1. Allow for reviewing patterns of access
2. Allow for discovery of attempts to bypass system controls
3. Allow for discovery of use of privilege
4. Act as a deterrent
5. Provide additional assurance

Auditing should provide a database SSO (DBSSO)[1] with a manageable set of data that can be analyzed to detect where in the AIS security scheme violations have taken place and by whom [Gluck 93]. Auditing is used to detect and deter penetration of an AIS and to reveal instances of misuse [Audit 88]. Audit serves as a deterrent because it documents and adds accountability to user actions which the user can be forced to explain. With the results of this data, an AIS security mechanism can be adjusted to plug ‘‘leaks” that are identified. Auditing allows for the recording and review of all security relevant events and provides an additional level of user assurance that misuse will not go undetected [Seiden 87]. Audit can also verify that the system is intact and working effectively [Filsinger 93].

The computer security research community has sought to further refine auditing characteristics. Jajodia et al., aim for a complete reconstruction of all events in the DBMS, specifically who has been accessing what data in what order [Jajodia 89, 90; Kogan 91]. Hosmer proposes flexibility to match the threat and the needs for analysis [Hosmer 90]. Williams and LaPadula aim for external consistency [Williams 93]. Filsinger aims for both internal and external integrity [Filsinger 93]. The developers of intrusion detection and countermeasure tools seek from auditing evidence of activity known to represent misuse and the identification of user activity discernibly anomalous from historical usage [Halme 95]. Finally, Schaefer et al., go to lengths to examine modes of access related to audit [Schaefer 89b].

2.3 SECURE DBMS vs. SECURE OS AUDITING

The TCSEC audit requirements for a secure OS also serves as the basis for secure DBMS audit. However, there are some differences in applying auditing requirements to a secure DBMS. In some cases, the differences are due to fundamental variations in the roles of a DBMS versus an OS. In other cases, the distinction is due to the incorporation of features that, while not fundamental to the

1.The document uses ‘‘DBSSO” to refer to user/roles that have the appropriate privilege to access the audit mechanism. Note that the TCSEC requirement for such a person/role is not introduced until B3.

DBMS paradigm and not required for OSs, have become de facto requirements of DBMSs. This section describes the differences between DBMSs and OSs and how they impact auditing.

2.3.1 Object Differences

There is a greater variety of object types in a DBMS than in an OS. The typical object in an OS is a file or segment. In a DBMS, there can be a relation (table), a tuple (row), metadata, an index, and others. The TCSEC requires the audit of objects introduced into the user's address space. The TCSEC defines two classes of objects: named objects, which are generally used for discretionary access control (DAC), and storage objects, which are generally used for mandatory access control (MAC). For auditing, the TCSEC makes no distinction between these two classes of objects.

For many MLS DBMSs, the object of MAC (the storage object) is a row or tuple. But the named object may be a tuple, view, or relation. For example, Trusted Oracle storage objects are tuples and pipe messages. Whereas Trusted Oracle named objects are tables, views, sequences, program units, and snapshots [Oracle 94]. Thus, while there may be some entities that are both named and storage objects, some DBMS objects are only named objects or only storage objects. Since the TCSEC makes no distinction regarding the auditing of named and storage objects, the introduction of both named and storage objects into a user area must be audited.

One unique aspect of DBMS objects is that in some instances they tend to overlap (e.g., views). This overlap can be a problem from an audit perspective if the auditing is done at the wrong level of granularity. For example, two overlapping views may share a common set of data tuples, but the access permissions on the two views may be different. If the DBMS provides only tuple-level auditing, then it is not possible to capture the use of the view or audit whether or not it was a legitimate access. The twofold solution to the problem is to have a system flexible enough to provide auditing at both the view and tuple level, and to ensure that the type of access employed (via the view or via the base tables) impacts the type of auditing provided [Schaefer 89a].

The greater number of DBMS objects means that more data objects will be accessed. If the audit mechanism audits each object accessed, this could greatly increase the size of the audit trail. Given that the number of DBMS objects can potentially be several orders of magnitude greater than the number of OS objects, the impact on the audit trail size could be extremely significant. Such an increase could render meaningful interpretation of the audit trail impossible. Potential solutions to this problem include the development of highly sophisticated audit reduction tools (see Section 5).

2.3.2 Queries and Transactions

Data accesses are fairly straightforward in an OS since DAC is done only on the object itself. Thus, no matter what path you take to get to an OS object you are subject to that object's DAC controls. In a DBMS there may be numerous access paths that can access the same data, each of which has its own separate DAC controls. This is because of views, and the fact that in most DBMS implementations gaining access to a view implies access to all the information included in the view, regardless of the DAC controls on the underlying base tables or views. In a DBMS, the determination as to whether access is granted or denied is based on the choice of access paths (e.g., through certain views or direct access to the base tables). The determination of which access path to employ is at least partially under user control. Therefore, the access path that the user intended to employ may be reflected in the user's query.

The increased sophistication and power of query processing in a DBMS greatly complicates user access and the associated role of auditing. Many operations supported by relational DBMSs can be divided into suboperations. In addition, many of the DBMS operations can also can have a propagating effect. That is, one macro operation may cause, or force, other macro operations to take place. As a result, there are a far greater number of potential audit events. Not only does this increase the size of the audit trail, but it also increases the number of configuration options for audit collection/reduction.

Another distinction between DBMSs and OSs is the concept of transactions and transaction management that applies to DBMSs but not OSs. A transaction is a grouping of statements within a database application that form a logical work unit. By definition, a transaction must leave a database in a consistent state. If a transaction fails during its execution (e.g., because one of the comprising statements cannot obtain a lock of a DBMS resource), the transaction will abort, and those actions that were performed by the transaction up to that point must be undone (rolled back). In some instances, the transaction will automatically repeat at various subsequent time intervals until it finally succeeds. If these subsequent transactions also fail to complete, those transactions will also be aborted and rolled back as well. Once again we see potential for a significant increase in the number of auditable events, and in the size of the audit trail.

2.3.3 DBMS Integrity

One of the features found in most relational DBMSs is support for integrity policies. The integrity policies enforced by DBMSs are more than the label-based concept of integrity reflected in [Biba 77]. Rather, integrity in the DBMS sense includes concepts of correctness (e.g., value integrity) and consistency (e.g., referential integrity). Neither the TCSEC nor the TDI consider incorporating such integrity concepts in their definition of security. Enforcement of referential and entity integrity is an aspect of DBMSs that makes them unique from OSs. The point that needs to be noted here is that auditing of DBMS integrity events, which is not required by the TDI, could greatly increase the size and complexity of an audit trail. For additional relevant work, consult the associated TDI companion document *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96].

Given the additional complexities of a DBMS, including finer object granularity, support for views, queries, transactions (including rollback), and integrity policies (and possible auditing thereof), the complexity of auditing in a secure DBMS may greatly exceed that of a secure OS.

2.4 PROBLEMS AND ISSUES

As has been observed through the previous discussions, DBMS auditing poses many serious issues. Table 2.1 offers pointers to where some of these issues are discussed within this document. Secure DBMS vendors have implemented audit mechanisms and have addressed many of the issues raised above. Approaches from the two DBMSs that have completed NCSC evaluation (Oracle and INFORMIX) are cited throughout the document to provide examples of NCSC acceptable implementations.

Topic	Issues	References
Audit Policy	What events should be audited? How should an appropriate audit policy for the perceived risks be determined and adapted when the threat changes?	• Discussion, ¶ 3.1 • Named and stored objects, ¶ 2.3.1 • The TCB boundary, ¶ 3.31 • Use of OS audit trail, ¶ 4.1.3 • Class B2 covert channel requirement, ¶ 5.6
Auditable Events	What events should the DMBS be capable of recording? What data about each event should be recorded?	• Discussion, ¶ 3 • Satisfying requirements, ¶ 4.1.1 • Audit settings, ¶ 4.2 • INFORN Online Secure events in Appendix
Audit Storage	If the level of granularity of audit within a DBMS is too fine, the resulting volume of audit records generated will be too high to permit effective identification of security-relevant events. Will denial of service occur if the DBMS shuts down when the audit trail overflows or the audit mechanism malfunctions?	• Discussions, ¶ 3.2, 4.3. • Storage of DBMS audit records in OS audit trail, ¶ 4.1.3 • View-based access events, ¶ 4.2.1 • NPM architecture impact, ¶ 6.4.1 • Data recording flexibility, ¶ 7
Audit Creditabilit y & Protection	Is the audit mechanism credible? How is the DBMS audit mechanism/guarded against disclosure, alteration, purging or disabling?	• Discussion, ¶ 4.5 • DBMS protection, ¶ 4.1.3 • Protecting against intruders, ¶ 4.3.1e
Audit Analysis	Retrieving an item of interest is quite difficult given the sheer volume of audit records produced when DBMS auditing is enabled. Should the audit trail be stored is database itself to facilitate analysis?	• Discussion, ¶ 5 • The need for tools, ¶ 4.1.2 • Centralized audit trail, ¶ 4.3.1a&e • Distributed audit trail, ¶ 4.3.1b • OS-based analysis, ¶ 4.3.1c • Tool performance, ¶ 4.3.2 • Summary of analysis issues and current capabilities, ¶ 8.

Table 2.1: Roadmap to Discussions on DBMS Security Dilemmas

SECTION 3

AUDITABLE EVENTS

Audit can be performed on either a continuous basis, or triggered by a particular events [Albert 92]. Audit, in an accounting sense, for fraud is continuous. In contrast, triggered auditing only takes place when a security relevant event is deemed to occur and the need for security auditing is elevated (e.g., to gather evidence and log the damage so it can be repaired). This section opens with a description of what an audit policy covers and its variables. Following this is a discussion of each variable, introducing the actions to audit, objects to audit, and the data to be collected about each event. This section should answer the questions, what could you and should you audit.

3.1 AUDIT POLICY

An audit policy is a statement of high-level rules, goals, and practices that describe how the organization collects, manages, and protects its audit data [Filsinger 93]. Audit policies impose a structure on DBMS auditing encompassing technical, administrative, and procedural aspects of DBMS audit. The technical aspects of the audit policy define which objects and events to audit. The policy should be sufficiently detailed that it can be implemented with the technical capabilities of the target DBMS.

The audit policy can help to delineate which audit options are to be used: part, some, or all of the time. Once it has been determined what objects are to be controlled and which events are security relevant, then the DBMS audit options can be set to target the objects of interest and optimize the audit resources. The audit policy may vary depending on the level of risk and the perceived threats. For example, a policy of auditing only modifications to data may be changed to another which audits both access and modification [Hosmer 94]. The DBSSO normally establishes these policies using risk management principles, so that there is more auditing where there is more risk.

3.2 SELECTIVE AUDITING

Selectable audit events can be provided by the DBMS as a way to balance the trade-off between too much audit data against too little or none. Historically, audit options have been relatively inflexible, requiring DBSSOs to enable large amounts of auditing even if their audit policy only warranted a small amount of audit activity. Extensive auditing can degrade performance in AISs not designed to handle such activity and the resulting audit trail space requirements can exceed capacity. Selective auditing may be the only viable alternative to manage rapid DBMS audit trail growth.

Extreme flexibility should be the hallmark of a good DBMS audit mechanism, allowing the DBMS owner to tradeoff performance, storage capacity, security, and mission needs. Audit options enable the DBSSO to maintain performance, conserve space, and to focus on the significant audit data. DBMS audit mechanisms should provide a highly configurable set of audit capabilities to ensure that the data captured is no more or less than necessary. For example, some DBMSs provide the ability to audit over 200 separate database events [Oracle 94a].

Different users, groups, and objects in a system may have different audit requirements. For example, a DBSSO may want to single out a particular user or group of users and gather extensive data. Similarly, a highly sensitive view of a database may require a significantly higher level of audit than other views. Audit events can be divided into groups with similar types. Albert et al., divide database audit events into several different classes as shown in Table 3.1 [Albert 92].

Event Types	Data Included in Audit Trail
End User	All actions taken by a DBMS on the part of the end-user (includes SQL commands)
Database Admin	Actions by operators and database administrators to control or configure DBMS operation
Database Security Admin	Granting and revocation of privileges and permissions, and setting labels
Database Metadata	Operations relating to the structure of the database
Database System Level	Utility commands, deadlock detection, rollback and recovery, etc.
Opening System Interface	Utilities used and configuration changes performed between the OS and the DBMS
Application	Application specific security relevant events supplied by the application

Table 3.1: Representative Audit Trail Event Types

Enabling and disabling of audit options can be implemented in different ways. Oracle has the AUDIT SQL command to turn on/off audit options [Oracle 94b]. INFORMIX uses audit masks to specify a set of user events to be audited [INFORMIX 94]. Audit masks are implemented as a sequence of bits, one for each predetermined auditable event. The following describes the type of audit masks a database security administrator (DBSA) can use:

- Compulsory Mask - Events which are always audited for all users
- Individual User Mask - Events for a particular user that need to be audited
- Default Mask - Events which are audited for those users without an individual audit mask
- Template Mask - A pre-defined set of auditable events that may be used to quickly change another audit mask
- DBSA Mask - Events which are audited for all DBSAs

In the INFORMIX implementation, each individual user always has the compulsory and individual (or default) mask applied to their actions. The DBSA uses a trusted interface tool, SAFE, to manage the audit masks. The compulsory, default, and DBSA audit masks cannot be deleted. Audit mask changes take effect immediately upon changes to a user’s individual mask.

3.2.1. Audit Event Requirements

Systems offering higher trust include more auditable events [Hosmer 94]. For example, lower level systems provide passive auditing, whereas high level systems can aid in proactively anticipating problems. Security relevant events are identified by the vendor and may be audited for success, for failure, or for both. According to the *Guide to Understanding Audit in Trusted Systems* (Audit 88], depending on the TCSEC evaluation class of a system, the events in Table 3.2 should be auditable.

TCSE Class	Required Audit Events
C2 and up	• Use of identification and authentication mechanisms • The introduction to and deletion of objects from a user's address space • Actions taken by privileged users (e.g., operators and DBSSOs) • Production of printed output • All (other) ''security-relevant'' events
B1 and up	• Actions taken to disable or override human readable output labels • Actions taken to change sensitivity ranges or levels of I/O devices and communication channels
B2 and up	• Events that may exercise covert channel channels
B3 and up	• Events that may indicate an imminent violation of the system's security policy

Table 3.2: TCSEC Required Audit Events

In addition to the ''security-relevant" events identified in Section 5 of the Audit Guideline, the TCSEC has been interpreted for the auditing of unadvertised TCB interfaces [Interp 95]. Interpretation 1-0286 was effective as of 1994-04-18 and applies to classes C2, B1, B2, B3, and A1:

> The following interprets the requirement that ''The TCB shall be able to record the following types of events:...and other security-relevant events".
>
> The TCB shall be capable of auditing all security-relevant events that can occur during the operation of the evaluated configuration, including events that may result from the use of TCB interfaces not advertised for general use.

The TDI requires that the following events also be auditable [TDI 91]

TDI Class	Required Audit Events
All	• All access control decisions
None	• Tuples not returned because they do not satisfy the query

Table 3.3: TDI Required Audit Events

An appropriate set of security relevant events is not always easy to determine for a DBMS. The TDI notes that if each of several subsets (e.g., OS, DBMS) meets the audit requirements locally, then the set of audit records generated will meet the requirements for the whole system. If not all the TCB subsets meet the audit requirements locally, then it must be demonstrated that the audit requirements for the whole system are met by the cooperative action of the set of TCB subsets. For the DBMS, auditable events are the individual operations initiated by untrusted subjects (e.g., UPDATE, INSERT, DELETE), not just the invocation of the DBMS. Individual operations performed by the DBMS TCB subset, if totally transparent to the user, need not be audited by the DBMS. As an example, INFORMIX-OnLine/Secure auditable events are provided in the Appendix.

3.2.2 Audit Event Data

As observed previously, ‘‘DBMS vendors have recognized that auditing requirements vary a great deal from site to site based on application and environment” [Schaefer 89b]. Table 3.4 presents the minimum TCSEC requirements for what data should be captured for each auditable event [DoD 85].

TCSE Class	Required Data
C2 and up	• Date and time of event. • User who caused the event to occur. • Type of event. • Success or failure of the event. • For I&A events, the origin of the request shall be included. • For events that introduce an object into the user’s address space and for object deletion events, the name of the object shall be included.
B1 and up	• The object’s security level.

Table 3.4: Audit Record Content

The TCSEC has been interpreted with regard to the data collected following an unsuccessful login attempt [Interp 95]. Interpretation 1-0006 was effective as of 1993-10-20 and applies to classes C2, BI, B2, B3, and Al:

> The following interprets the requirement the ‘‘The TCB shall be able to record the following types of events: use of identification and authentication mechanisms,... For each recorded event, the audit record shall identify: data and time of the event, user, type of event, and success or failure of the event.”
>
> > While the audit mechanism is requirement to be capable of producing a record of each login attempt, on failed login attempts it is not required to record in the audit record the character string supplied as the user identity.

Over and above the TCSEC requirements, the specific data that should be collected for each event depends both on the nature of the event and the needs of the auditor. This includes not only what to audit, but specifically what data should be collected for those events which are audited. Not only does a DBMS need to provide flexibility as to which events are to be audited, but what data is to be collected for each event.

3.3 AUDITING ACCESS CONTROL EVENTS

The TCSEC has been interpreted for audit of access control events [Interp 95]. Interpretation 1-0073 was effective as of 1993-10-20 and applies to classes C2, B1, B2, B3, and Al:

> The following interprets the requirement that ”The TCB shall be able to record the following types of events:..introduction of object’s into a user’s address space (e.g., file open, program initiation). . .“

Auditing the attempted introduction of an object at the point of passing the security access control checks satisfies the above requirement, even though the object may not actually be introduced into the subject's address space because of failing later checks not related to security.

This section considers how best to determine what should be considered an auditable DBMS access, including ramifications for context and content dependent accesses.

3.3.1 What is a DBMS Access?

Consider the situation in which the audit requirement is that *all* accesses must be audited. Let us use as our example a query issued against a non-indexed relation (such as the relation in Figure 3.1). Because the relation is non-indexed, the DBMS will retrieve each tuple in the relation to see if it satisfies the logical conditions of the query. Now the question is, do each of these retrievals constitute an access that needs to be audited? If so, then the entire relation (one tuple at a time) will be accessed and the relevant data placed in the audit trail. The consequence of such an action is a huge, and to a large extent meaningless, audit trail.

Name	Job Title	Age	DEPT	Security Level
John Smith	Engineer	21	2	U
Mary Jones	Manager	29	4	U
Jeff Johnson	Spy	32	13	TS

Figure 3.1 EMPLOYEE Relation

What then should constitute an auditable DBMS access? The answer most consistent with the TDI and TCSEC is that an auditable access is the data that is passed beyond the TCB boundary. A consequence of this interpretation is that if the TCB encompasses the entire DBMS, which is true for some Class C2-B1 DBMSs, then only the data that is returned to the user is considered an access. The major problem with this approach is that data not returned but used as a basis of the selection and qualification process is not represented. This would adversely impact detection of unauthorized access attempts including potentially the detection of the exploitation of covert channels at B2 and higher.

The most logical answer to this dilemma would be to consider only that data which is returned after the DBMS performs a logical qualification on the fetched tuples as an auditable access. But if the logical qualification function is part of the DBMS TCB, and it in turn passes the data to still another part of the TCB, then one is in effect auditing the internal functions of the TCB. Such an action is not required by the TDI and the TCSEC.

The truth is that there is no single best answer to the question of what constitutes a DBMS access. What is apparent is that the answer is dependent upon the nature and structure of a given DBMS TCB and hence is architecturally specific. This question of auditing DBMS accesses is explored further in Section 6.3.1. However a DBMS access is identified, DBMS vendors need to define it in their assurance documentation.

3.3.2 Content and Context Dependent Access Control

DBMSs can enforce content-dependent or context-dependent access control policies. The most common implementation of these policies involves views and triggers. For example, a view might be constructed against the EMPLOYEES relation (Figure 3.1) and would constrain some users to access only those tuples in which the Employees were in Departments 2 and 4. Thus, any changes to the contents of these fields through the new view would have impact on subsequent user access requests. At the very least, this construction would increase the size of the audit trail because one more auditable event would be included.

More sophisticated DBMSs can also support context-dependent access control. For example, a trigger could be set on relation X that would prohibit a user from accessing the relation if he has reviewed specified tuples in relation Y or Z in the last 24 hours. Such a technique might be used to address inference threats. To determine whether the access control condition is satisfied, a history log would be required. A separate history log could be developed for this purpose. In fact, a system such as this is employed in the LOCK DBMS [Haigh 90]. In order for the audit mechanism to detect potential security violations of such a policy, it would need to collect all accesses to relations Y and Z by all users. This action would likely significantly increase the size of the audit trail.

3.4 AUDITING DBMS INTEGRITY

Most DBMSs and MLS DBMSs support referential integrity. That is, the DBMSs support mechanisms for placing bindings or constraints on multiple relations. As noted in Section 2.3.3, the TDI does not require auditing of DBMS integrity events. However, since integrity enforcement is such a key feature of a DBMS, Jajodia believes that integrity related audit events should be a consideration [Jajodia 90].

A simple example of this is the definition of a foreign key in one relation that refers to another relation. Consider a new multilevel relation, DEPT, which has three attributes: DEPTNO, NAME, EMP_COUNT and a Row_Label for each tuple. The EMPLOYEES relation, defined earlier, also has a DEPTNO field, and this field could be identified as a foreign key for DEPT..

DEPTNO	**NAME**	**EMP_COUNT**	**ROW_Label**
2	John Smith	21	U
4	Mary Jones	29	U
13	Jeff Johnson	32	TS

Figure 3.2: DEPT Relation

Now, a database designer might wish to use referential integrity to ensure that each employee must have a valid department number in the EMPLOYEE relation. During the completion of a tuple insertion into the EMPLOYEE relation, the DBMS checks the DEPT relation for the existence of a DEPT record that has the same value for DEPTNO as the one being inserted. Not only does this extra operation increase the potential for the generation of an audited event, but the nature of these new audit events could take many forms. For instance, if a No Valid Department error is returned, then the New Employee record does not satisfy the integrity constraint, the insertion operation fails, and it should probably be audited. On the other hand, if a valid department record exists but is classified at a level higher than the user's operating level, then the successful completion of the insertion may be considered a write-down of the fact of the existence of the department. Likewise, the failure of the insertion due to classification would probably also be an event that should be audited.

Another more complex example of referential integrity occurs when an operational constraint is placed on a pair of relations. For instance, a database designer may wish to impose a constraint where employee records cannot exist without a Valid Department. One way of doing this is to define a delete cascade on the DEPTNO foreign key. A cascading delete operation will force a DBMS to delete all EMPLOYEE records with the corresponding value of DEPTNO when a given DEPT record is deleted, and such an operation can potentially generate many auditable events.

SECTION 4

AUDIT STORAGE, PERFORMANCE, AND PROTECTION

The previous section described what events could be audited and how to decide which events should be audited. This section presents an implementation oriented view of auditing. Specifically, it addresses for specific data that can be collected (Section 4.1), when to collect it (Section 4.2), where to store it (Section 4.3), how much to store (Section 4.4), and how to protect it (Section 4.5).

4.1 AUDIT IMPLEMENTATION

There are many goals for audit implementations. The most basic is to simply meet the TCSEC requirements. However, researchers have also proposed audit mechanisms that capture the intent of users. Other audit mechanisms allow applications to insert audit entries. Issues related to these approaches are discussed in the following sections.

4.1.1 Auditing to Satisfy the Requirements

Section 3 discussed the audit requirements from the TCSEC. Given these requirements, they should be relatively straightforward to satisfy. Auditing of I&A events is normally done at a single point within the TCB. Auditing for access control events depends on how objects are stored and the granularity of the access control mechanism.

For DAC, controls are typically provided down to the table or view level. Thus, each access to any or all tuples within a table or view will require a single audit record to be generated to satisfy the auditing requirement for DAC. However, for MAC, it can be a very different story. MAC is typically provided down to the tuple level, and can even be provided down to the element level. The number of MAC decisions that are made depends on how tuples are stored internally, not necessarily how they are represented to the users. For example, a DBMS can provide tuple level labeling by storing all tuples of like sensitivity levels in a single object (e.g., INFORMIX OnLine Secure [INFORMIX 94]). If this is done, then a single MAC check is done for each level, rather than one per tuple. This may significantly reduce the amount of audit data that must be recorded to satisfy the requirements.

If tuples are individually stored and labeled, then each time a MAC check is made on an individual tuple, that data must be recorded in the audit trail (for enabled events). This can cause a significant amount of data to be recorded in the audit trail, depending on the query, contents of the database, and its organization. For example, a query by an Unclassified user which asks to return all tuples from a table will require the access decisions to each and every tuple in the table to be recorded in the audit trail, since a MAC check will have to be done on each tuple to determine whether it can be returned to the user. Now, according to the TDI, if the user includes a WHERE clause in the query, then all tuples that are excluded because they do not satisfy the selection criteria do not have to be audited.

4.1.2 Auditing to Capture Intent

If only the security relevant data explicitly required by the TCSEC is recorded, it may be difficult for a DBSSO to grasp the intent of a user's query from this data. This difficulty may be compounded based on where during query processing the auditing is actually performed. These two issues are discussed in the following paragraphs.

To help grasp the user's intent, it is desirable to record the entire text of each SQL statement in the audit trail. This allows the DBSSO to better understand what a user was trying to do, rather than trying to infer this based on the audit record generated during the processing of the query. Recording transaction starts and ends will also help the DBSSO understand what queries or changes are being attempted and which changes are actually committed to the database.

In high assurance DBMSs, the parser, compiler, and optimizer may have to be excluded from the DBMS TCB to reduce the size and complexity of the TCB. If this is the case, then the TCB may receive only correctly formed and optimized queries rather than the raw original queries from the user. When trying to determine the intent, it may be much more helpful to see the original query, rather than the parsed and optimized query. As such, it would be helpful if the original query submitted by the user was recorded, rather than the optimized one. To do this with assurance, there must be an extension of the TCB which captures user input at the user interface, audits this data, and then passes this data on to the parser/optimizer/compiler before it is eventually submitted back to the TCB.

Thus, for all operations:

- The entire text of the SQL statement should be recorded.

In addition, for certain operations, it is sometimes desirable to collect additional data beyond what is minimally required. Therefore:

- For UPDATES, the old and/or new data values should be recorded.
- For INSERTS, the value of the inserted data should be recorded.
- For DELETES, the value of the deleted data should be recorded.

Other data that might be useful to record for failed events is the reason why the event failed. Related to this is recording the security levels or authorizations associated with the user at the time of the event or a means for obtaining this data. This data can be helpful when trying to identify why an access attempt failed.

Regardless of when a query is recorded in the audit trail, the audit trail must ensure that the DBSSO can trace each piece of audit data back to the query which caused that data to be included in the audit trail. This is crucial because a single query can cause a huge amount of data to be recorded in the audit trail. This can be done in a number of ways, including storing all related data for a query in a single audit record, or including some type of reference number with each audit record that allows it to be traced back to the query which caused the record to be generated.

Similar to associating data with queries, it would also be helpful to be able to easily associate queries with transactions. Each transaction is composed of a series of queries. From the auditing perspective, all of the accesses generated as a result of a transaction constitute a legitimate read and write. If all audit records are written out, whether a transaction is rolled back or not, not only are additional audit events generated by the rolling back and repeating of transactions, but many of these audit events provide an essentially false picture of the state of the system. In addition, the repetitive attempts at access may be logged by the audit mechanism and potentially misinterpreted by the DBSSO as user-attempted security violations.

To address these problems, the audit mechanism must have some way to identify those events that are part of a completed transaction and those that are not. This identification may require the transaction manager to notify the DBMS when it begins, completes, or fails a transaction. The appropriate data could then be appended to the records in the audit log corresponding to the data that was accessed. The difficulty with this approach is that the data about accesses will likely be recorded prior to the data regarding transaction completion or failure. To make the audit data useful to the DBSSO some form of audit analysis tool is needed to correlate the data [Schaefer 89a]. To help a DBSSO trace the progress of a transaction, it would be helpful if the audit analysis tool could thread the associated queries together.

Therefore, the following data is recommended for capture in the audit trail:

- Reason why an event failed, including the user's security levels or authorizations, or a pointer to this data.
- The original query, prior to parsing or optimization.
- Each audit event record should be traceable back to the query which caused it.
- Each query that is part of a transaction should be traceable to the transaction.
- Whether a transaction eventually was committed or was rolled back.

4.1.3 Application Specific Auditing

Just as most trusted OSs enable their applications to insert audit records into the OS audit trail, it is recommended that DBMSs also provide their applications with a similar interface. The goal is to collect audit data that is meaningful to the DBSSO. For example, Compartmented Mode Workstation (CMW) requirements include the ability to accept data from processes which have been given the privilege (e.g., writeaudit) of writing their own audit records [Picciotto 87]. Another privilege (e.g., suspendaudit) can then be used by these processes to suspend and resume OS auditing of their actions. If the relevant application is collecting event data, then the underlying OS can have its lower level auditing disabled for activities related to that application. Thus, the OS can avoid cluttering the audit trail with low level events and retain the focus on activities of the subjects at the proper level.

This capability can be provided for DBMS applications in a couple of ways. If the DBMS inserts its records into the OS audit trail, then applications can use this same capability to insert records into the OS audit trail. This is what INFORMIX [INFORMIX 94] and Oracle do when the database is set to write all audit records to the OS audit trail, rather than storing it in a DBMS audit trail [Oracle 94b]. If the DBMS keeps its own audit trail, it should also have a mechanism for privileged database applications to insert their own audit records into the database audit trail and provide appropriate protection of the audit trail.

Another capability for generating ad hoc records that can provide additional audit flexibility is the ability to use triggers to generate customized audit records [Oracle 94a][2]. This allows the DBSSO to capture specific data based on whatever data the trigger can test on. For example, a trigger could be used during an UPDATE to capture both the old and new values of any tuple modified by the

2. Note that triggers were not considered in the Oracle evaluation [Oracle 94b].

UPDATE. Triggers could also capture different data based on tuple accessed, user performing the operation, time of day, object sensitivity, old or new element values, etc. Such a capability could be used to provide much of the desired collection flexibility described in Section 7. This capability also has the additional advantage that it places the trust of performing the audit into the DBMS audit mechanism, rather than having it done in an outside application. This helps minimize the amount of trusted code that must be developed to support an application specific audit policy [Hosmer 94].

4.2 AUDIT SETTINGS

Section 3 talked about various types of auditing that could be performed, including statement, privilege, and object auditing. What is to be audited, and hence what is recorded when an event occurs, is typically defined by the DBA, DBSSO, or object owner. For statements and privileges, audit parameters are defined for each statement or privilege. Whenever a statement or privilege is used, the appropriate data is collected, regardless of the audit settings of other statements or privileges, or the objects accessed.

For views and base tables, the data to be recorded is not necessarily straightforward due to the explicitly defined relationships between them. Typically, audit setting data is recorded for each:

- Base table definition,
- View definition.

Based on this, audit is typically performed each time a view or underlying base table is accessed. The exact details of what is audited depends on the defined precedence between view and base table audit settings and how these settings are established. These two issues are discussed in the following subsections.

4.2.1 Audit Setting Precedence for Views and Base Tables

The DBMS designer must decide what the relationship is between audit settings associated with a view and the settings of any base tables or views accessed by the view. A number of options are available:

1. The highest level (most abstract) view settings take precedence.
2. The base table settings take precedence.
3. A union of the settings of all views and base tables accessed is used.
4. The intersection of the settings of all views and base tables accessed is used.

These options each have certain benefits and drawbacks as follows:

Using the highest level view settings is a simple mechanism and allows the creator of the view to specifically identify what is audited and when. However, this may override the desires of the owners of the underlying views or base tables. In fact, with this mechanism, changes to the audit settings of underlying views or base tables will have no effect on higher level views, unless changes are somehow propagated to other views.

Using base table settings ensures that the owner of the base table has complete control over what is audited when that table is accessed no matter how it is accessed. This has the benefit that auditing will be consistent across all views to the underlying base table but has the disadvantage that auditing cannot be tailored for specific views. In this approach, changes to the audit settings of a base table would automatically be used by all views which access the table, and view level settings would be unnecessary.

Using the union of the settings would be a more complicated mechanism. Given that a view can build upon other views which in turn can be built on other views, etc., the number of settings that must be UNIONed together could be difficult to compute. This type of mechanism could cause the amount of audit data generated to be much larger than what is desired if the audit settings are not managed properly, but it ensures that the auditing desires of the owner of each view or base table are met because the union of these settings is what is audited. This mechanism would have the advantage that changes to the audit settings of more primitive views or underlying base tables would automatically be invoked when higher level views are accessed. Trusted Oracle functions this way by generating a separate audit record for each view accessed by a particular function [Oracle 94b]. In this way, Oracle does not need to compute the UNION of the settings, rather it automatically generates a separate audit record for each view accessed by a function, regardless of how the user got to that view (i.e., directly or through another view). It generates each record based on the settings of that particular view and if these settings indicate that no record should be generated, then no record is created. This has the disadvantage that multiple redundant audit records may be generated but it ensures that all uses of a particular view are recorded.

Using the intersection of the audit settings (i.e., only audit when all audit settings indicate that it should be done) may be difficult or impossible to compute. Intersection may also cause the amount of audit data captured to be insufficient for reconstruction of the event which caused the audit record to be generated. Without proper management, the amount of data audited could be null. It is not clear how intersection would be computed when multiple views are used to construct a higher level view. It is not expected that this would be a reasonable choice for an audit precedence mechanism.

INFORMIX avoids this issue by supplying audit masks that are associated with individual users, independent of the objects they might access [INFORMIX 94]. These masks are used to indicate which events are to be audited for that individual (e.g., object creation, deletion, access; database or table privilege granting or revoking). Using this mechanism avoids the necessity to determine precedence of audit setting among views and base tables but does not provide the flexibility of adjusting audit settings on a per view or base table basis.

4.2.2 Establishing New Audit Settings for Objects

Once the audit settings precedence among views and base tables has been established, a specific procedure for establishing and maintaining audit settings needs to be defined by the DBMS vendor and implemented in the DBMS to promote a consistent and easy to manage audit capability. Since a base table is the most primitive view of that data in the database, the settings on a new base table are initially not related to any other view or base table. As such, the owner of the base table (or those with the appropriate privilege/role) should simply enable the audit settings as desired and that should complete the operation.

When a view is created which is based on the underlying base table, the settings can be defined in a number of ways, and should be affected by the audit setting's precedence defined between the view and the base table as discussed previously. If audit settings are only recorded for base tables (e.g., choice 2 from above options in Section 4.2.1) then no audit setting data needs to be recorded when a new view is created. If the union of the settings is used (choice 3) then the creator of the view simply adds any additional data that should be audited to this view definition. When the settings are UNIONed together, all the desired data is audited and, if the base table settings were subsequently changed, any new data would be captured. Data removed from the base table settings would not be recorded, unless it was also specified in the view definition that this data should be recorded as well.

If the highest level view settings take precedence (choice 1) then the underlying base table settings have no affect on what is audited. In this case, the user could create new audit settings from scratch, or for consistency or ease of use, the default settings of the new view could be set to the union of the underlying views or base tables accessed by the view (Schaefer 89b]. This would make the default settings equivalent to the settings on the underlying views. These default settings could then be modified as desired by the owner of the new view.

4.2.3 Modifying Audit Settings

Established audit settings typically evolve over time. The difficulty here is trying to make the audit settings as consistent and easy to manage as possible while providing the desired flexibility. The TCSEC has been interpreted to ensure that the enforcement of audit settings is consistent with the AIS's protection goals [Interp 95]. Interpretation 1-0004 was effective as of 1993-10-20 and applies to classes C2, B1, B2, B3, and Al:

> The following interprets the requirement the ‘‘The ADP system administrator shall be able to selectively audit...”
>
> > If the TCB supports the selection of events to be audited, it shall provide a method for immediate enforcement of a change in audit settings (e.g., to audit a specified user, to audit objects at a particular sensitivity level); however, the immediate method (e.g., shutting the system down) need not be the usual method for enforcement. The TFM shall describe both the usual enforcement of audit settings and, if the immediate enforcement method is different from the usual one, how an administrator can cause immediate enforcement.
> >
> > The TFM shall describe the consequences of changing an audit state dynamically if such changes could result in incomplete or misleading audit data.

If audit settings are only recorded for base tables (choice 2), then changes to the base table audit settings are automatically used whenever they are accessed by a view. If audit settings are the union of all views and base tables accessed by the view (choice 3), then any changes to the audit settings of the underlying views or base tables are also automatically reflected in the auditing performed by any higher level views.

Only when the audit settings of the highest level view take precedence (choice l) do changes to the audit settings of more primitive views or base tables not affect the audit records generated by higher level views. Mechanisms could be created which aid the user in propagating changes to views which reference a more primitive view. This is a research issue. Difficulties could arise in finding these view definitions and the user may not possess the appropriate permissions to change higher level views which the user may not own.

4.3 AUDIT STORAGE

Previous sections discussed what events should be auditable and what data should be recordable for each event. This section discusses storing the audit data that is ultimately captured by the DBMS.

4.3.1 Audit Storage Location

A DBMS (particularly the server portion) typically runs on top of a trusted OS which is performing its own auditing. The relationship between the OS and DBMS is one factor in the equation of how to store audit data most effectively. There are many other factors that affect the decision as to which storage option should be selected for storing DBMS audit data. Some of the possible storage options are as follows:

a) Store all DBMS audit data in a single audit trail that is independent of the OS audit trail (for MLS DBMSs, this would be a database-high object).

b) For MLS DBMSs, create a separate audit trail for each sensitivity level and compartment combination.

c) For DBMSs running on top of an OS, insert the DBMS audit records into the OS audit trail.

d) Merge the DBMS audit trail with or use the DBMS transaction/recovery log as the audit repository.

e) Forward the audit trail to a specified location.

These options each have certain benefits and drawbacks, some of which are described as follows:

a) Using a Single DBMS Audit Trail

The most common audit storage method is to store all the DBMS audit data in a single audit trail. This can either be within an audit database or a raw file managed by the OS. This method is fairly simple to implement as the DBMS owns and controls all the necessary resources. A single audit trail also aids the audit analysis task since all the audit data is chronologically stored in a single location, which makes it easier for an analysis tool to search for and correlate related events. The ease of analysis depends on the sophistication of the supplied audit analysis tools, which can range from the primitive (e.g., UNIX grep) to the sophisticated. For MLS DBMSs, all records in this audit trail can either be treated at system-high, or, if stored in a database, each audit record can be individually labeled to help track the session level of the user associated with each event. Labeling each individual audit record is beneficial because it provides another factor to distinguish audit records from one another. Labels also provide the ability to allow certain users to access portions of the audit trail without requiring a clearance for all levels of data stored in the audit trail.

b) One Audit Trail Per Sensitivity Level

Recording a separate audit trail at each sensitivity level is not commonly done, however, for certain MLS DBMS architectures, it is required unless the underlying OS permits blind write-ups. This approach is specifically required for DBMSs that are not trusted with respect to the underlying MLS OS's MAC policy. As described in Section 6.4, in the No MAC Privileges architecture, each DBMS instance logs audit records at its corresponding sensitivity level. Performing audit analysis can be more difficult in this configuration because the data is distributed across a number of different files. If the underlying OS permitted blind write ups, then a single audit trail could be created at the highest level of the database by allowing each component of the DBMS to append audit data to the audit trail. If this approach was taken, the lower level DBMSs would not be able to view the audit data that they contribute to the audit trail. Only a DBMS or a DBSSO with access to the highest level of data in the database would be permitted to view the audit data. No current DBMSs use the blind write up approach.

c) Merging the DBMS Audit Trail with the OS Audit Trail

Another option for recording audit trail data is to insert the DBMS audit trail data into the OS audit trail. This option depends on OS support for this capability. This has the advantage that the audit trails are merged into a single location. However, there may be difficulties in correlating related events between the OS and DBMS while ensuring that the chronology between events is maintained. Synchronized clocks will help alleviate the latter problem but the former depends on the granularity of data that is known and dealt with by the respective systems. For example, when a user who is accessing the OS then logs in to the DBMS, a record must be stored in the audit trail which associates the OS userid with the DBMS userid, or they must be identical on the two systems. This allows the actions of a single user to be correlated together regardless of whether the action occurred at the OS level or in the DBMS. However, even this simple approach can become complicated when in a client-server environment. Another difficulty factor is that an OS level audit analysis tool may not be able to access data stored in the database which is helpful to the analysis, whereas a DBMS specific audit tool may have that capability.

d) Merging the DBMS Audit Trail with the Transaction Log

Some discussion has been made in the literature (e.g., Schaefer 89b] about merging the DBMS audit trail with the DBMS transaction/recovery log since that data has to be recorded anyway. The idea is that it may be more efficient to record this data together since some of the data may be redundant between the two logs. However, the difficulties that arise have tended to outweigh the benefits of this approach. For example, dealing with roll backs and the subsequent loss of audit records caused by the rollback may be a problem, as well as trust issues surrounding the use of the transaction log and possible interference with the usability of the transaction log for its original purpose. Since the integrity of the audit trail must be guaranteed, merging it with the transaction/ recovery log requires that all software that creates, modifies, or purges the transaction/recovery log be trusted to protect the contents of the log and as well as ensure its accuracy.

However, using the transaction log as a supplement to a separate audit trail may not be a bad idea. The audit log can be used to record all the data required by the TCSEC and TDI, e.g., all access control decisions, I&A actions, etc., while the transaction log can then be used to provide additional data that might be helpful to a DBSSO when reconstructing events. The transaction log already provides data such as field values before and after changes, which changes are related to specific transactions, etc., and may be easily modified to include other additional data. In fact, the transaction log may be a more suitable place to collect some of the additional data that is recommended in Section 4.1.2. If the assurance of the transaction log is an issue, it may be fairly straightforward to provide a reasonable level of assurance as to its accuracy and integrity. The point within a DBMS where transaction entries are typically recorded is at a fairly low level within the DBMS. Thus, the mechanism should be fairly small and it should be feasible to provide reasonable assurance arguments as to its correctness. Some additional controls might be required to limit the ability to modify the transaction log to only a few select users and applications to ensure the integrity of the log after it is created.

e) Forwarding Audit Data

The ability to forward the audit trail to another location can be useful for a number of reasons [Schaen 91]. For example, if the DBMS is in a networked environment where an audit server is present, storing all the audit data on the server has the advantage that all the audit data is collected into a single location, making audit analysis and correlation easier. Obviously, a sophisticated audit server and audit analysis tool must exist which can store and manage this data. It must be able to correlate events across platforms and applications. From the DBMS point of view, all it needs to provide is the capability to forward audit data to an alternate location rather than storing it locally. Another reason might be to provide the ability to forward the audit data near real time to a separate location which is performing intrusion detection.

It is also desirable to be able to forward audit trail records to specific locations for protection reasons, such as write once devices, removable media, or duplicate locations [Hosmer 94]. These techniques are used to help protect audit trail data from intruders who might gain access to the system and wish to damage or destroy the audit trail to cover their tracks.

4.3.2 Audit Storage Format

Another issue that must be dealt with is how to store the audit data. Since audit trails typically tend to be quite large, efficiency of storage is a key issue. However, there are significant trade-offs between efficiency of storage and efficiency of analysis, since techniques to improve each capability tend to adversely affect the other. For example, minimizing the data recorded in each audit record by including references to other audit records or data available in the database (e.g., the identification database) rather than copies of the referenced data makes the audit trail smaller, but makes analysis slower because the references need to be looked up to find the data. Another problem with this is that sometimes the referenced data may not be available when the analysis is performed (e.g., due to audit trail truncation or if the analysis is performed on another system which does not have access to the auditing system's identification database). These audit problems are not specific to just DBMSs, they are the same as those faced by trusted OSs as well.

Another desire is to store DBMS audit data in a format that is compatible with the audit trails of other trusted components that will be used with the DBMS, such as the host OS or network. One could expect the DBMS to have a portable audit record format so that correlation of audit events between the DBMS and OS would be easier. Audit trail format standards efforts do exist [Sibert 88]. Some of these efforts, such as those by IEEE POSIX 1003.6, have considered including security extensions which would include a standard format for audit data as well as a standard Applications Programming Interface (API) for submitting audit data to the host OS.

4.3.3 Audit Storage Longevity

Another issue that can become very important when managing DBMS audit data is how long to keep the data [Audit 88]. The critical factors are how fast audit data is generated and how much storage is available to keep the data. There are a number of techniques for long term management of audit data. A single technique can be used, or multiple techniques can be combined in various ways to best optimize audit storage based on amount of storage available, retrieval time desired, data to be retained, etc. The primary techniques are described as follows:

<u>Archival</u> is the process of taking audit data from primary audit storage and moving it to some other storage location. This is typically to tape or some other secondary mass storage device. The oldest

audit data currently stored on the system is what is normally archived while the more recent data is left on the system in case immediate access is needed. Archival has the advantage that no audit data is lost but has the disadvantage that is it not immediately accessible if needed.

Truncation is basically discarding some portion of the audit trail, typically the oldest. Discarding can be directly from primary audit storage or from an archive. Truncation is typically inevitable because primary audit storage or secondary storage eventually becomes full. The decision to truncate is based on available storage, the criticality of the audit data, and its age, which obviously can vary widely from system to system.

Trimming is a technique whereby certain records in the audit trail are removed rather than chronologically truncating the audit trail. The records deleted should be selectable by the DBSSO. This allows the DBSSO to delete those records that provide more detail than desired or are deemed not important while retaining those records that are important to them. This has the advantage that the audit trail size is reduced while the important records are retained for future examination.

Compression can be used to reduce the physical size of the audit trail to a significant degree, sometimes as much as 50% or more [Sibert 88]. This has the advantage that less storage is required and no audit data is lost. The disadvantage is that a small amount of extra time is required to store or access the data (this is usually more than offset by the speed gain due to less disk/tape access).

Abstraction is used to retain statistical data about a series of audit records rather than the records themselves. This technique can be used to summarize data into a more compact form through use of counts, averages, etc. This format is sometimes more useful to systems such as intrusion detection systems who detect violations based on changes to ‘normal’ behavior, in addition to the reduction in audit trail size.

All of these techniques should be made available to the DBSSO in order for him to be able to select the combination of techniques that are most effective for the site’s particular needs.

4.4 PERFORMANCE

Performance has always been a significant issue for trusted OSs which perform auditing. Performance is even more critical to trusted databases because of the granularity of the objects involved and the potential volume of transactions to be audited. However, the techniques and issues surrounding audit’s affect on performance are basically the same as for trusted OSs and are not further elaborated here.

Due to the potential impact that audit can have on the performance of a high performance DBMS, it is much more critical for the DBMS developer to make the audit mechanism as efficient as possible. Regardless of what steps are taken to improve performance during auditing, one of the most effective mechanisms that can be supplied is to provide a highly configurable audit mechanism which permits selective auditing of particular events and selective recording of particular data for each event (See Section 7 for more detail on this capability). This allows site specific audit policies to be enforced in a manner that closely matches the site’s needs without recording unnecessary data and paying the performance penalty this might cause.

4.5 CREDIBILITY AND PROTECTION

For the audit trail to reflect a trustworthy record of the events which are supposed to be collected by the audit mechanism, it is necessary that the events which are recorded accurately and completely reflect events that occurred and that the audit trail is protected from modification. In addition to protecting the audit trail from modification, it is also important to control observation of the audit trail because it will typically contain sensitive data. Finally, it will be necessary to make decisions on what actions to take when the audit trail storage space reaches capacity. These decisions will typically involve tradeoffs between completeness of the audit record and availability of the system. Each of these issues is discussed below.

4.5.1 Audit Trail Credibility

The credibility of the audit log determines the degree to which the audit log can be used to reconstruct the history of the usage of the DBMS. This in turn affects the usefulness of the audit trail to identify attempted misuse, successful or not, of the DBMS, as well as the usefulness of the audit trail for investigation of misuse or intrusion detected by other means. Schaefer et al., note that the issue of the ability of a TCB to accurately record auditable events can be posed in two forms [Schaefer 89b]. A weaker form is to ask whether:

> When the TCB detects an event, is that event (and its association with a user and its time of occurrence) accurately recorded in the audit trail?

This question can be addressed by looking specifically at the audit mechanism and the protection afforded the audit trail. A stronger question is whether the event that is recorded represents what actually occurred. To answer this question one must do more than just examine the audit mechanism. The question can be divided into two parts. The first part is the question above. The second question is:

> When the TCB detects an event, (and its association with a user and its time of occurrence), did precisely that event occur during system execution?

This second question goes directly to the TCB's ability to control the actions of subjects on objects. If the TCB has no way to determine accurately that an event has occurred, it cannot mediate that event. The ramifications of this observation for different architectures is considered further in Section 6.

Access to the database through views is a particularly important case with respect to how much one can trust the relationship between what is recorded and what was executed. In particular, if the TCB detects a request to access the database through a view, then whether or not that event is the event executed may depend on the trustworthiness of the query language parsers or compilers. Recently, Schaefer et al., have taken a new look at what types of view based controls can be enforced with high assurance [Schaefer 94]. Their findings can be used to understand what can be trusted about the actual events executed when a given event is recorded in the audit trail.

Regardless of the degree to which the harder question of accurately reflecting actual event execution can be answered, it is essential that the audit trail at least accurately reflect detected events. Potential loss of audit data is one of the focuses of an evaluator's audit mechanism review. This means that assurance techniques appropriate for the target system assurance level must be employed to show that the audit mechanism accurately and completely records events given to it within the context of the current audit configuration as established by privileged users. When audit information is kept in database audit tables in the form of database rows, this information can only be lost or recovered as other database information is lost or recovered.

For example, Oracle table data can be lost if there is a media failure involving the Redo Log or the Control file (if the affected file is uniplexed) [Oracle 94b]. Otherwise, no relevant records can be lost because the audit trail entry is written before results are returned to the user. The user transaction does not have to commit before the audit record is written; writing the audit record is a sub-transaction that completes before the main transaction does. Therefore, the system could have audit records for events that rolled back and therefore never occurred, but would have records for all committed transactions and completed queries.

4.5.2 Access Control for the Audit Trail

Unauthorized modification or deletion of the audit trail can clearly subvert its usefulness for supporting any audit objectives. The TCSEC requires that [DoD 85]:

> The TCB shall be able to ... protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data.

Well publicized attacks on standard OSs have involved modification of the audit trail by the intruders in order to cover their attacks. In one important instance the attack was first noticed because the modification was done somewhat clumsily and actually caused the size of the audit trail to decrease [CMADIII 95]. Many studies have found that it would be desirable to do auditing at the level of applications which understand the meaning of a user's requests [Schaefer 89b]. Hence, it may be useful to allow applications to append data to an audit trail as described in Section 4.1.3. Nevertheless, it is essential that the ability to delete or modify records be closely controlled within the TCB.

While it is obvious that the ability to alter the audit trail must be tightly controlled, it is also the case that observation of the audit trail must be controlled. In general, audit records will directly or indirectly contain a significant amount of data about what is in the database. Hence, observation of the audit trail is a path for observing data from the database. In a system which supports MAC, audit trail data must be protected at a level which is an upper bound of the data recorded and the transaction causing the event. If the entire audit trail is protected as a single object, the audit trail must be protected at the highest level of any data in the database. In any system, observation of the audit trail should be limited to those with a legitimate need-to-know. Beyond the fact that unauthorized observation of the audit trail could compromise data which the DBMS is supposed to protect, observation of the audit trail directly reveals the setting of current audit parameters. This gives an intruder data which could be used to minimize the chance of detection of malicious actions.

There are several methods for providing the required protection for the audit trail. If the DBMS audit records are entered into the OS audit trail, the OS TCB will provide the required protections. However, the tools for correlating OS audit records with DBMS audit records and the procedures for granting privilege to the users who can review the audit trail must be carefully designed to make sure that they do not result in a loss or compromise of data in the audit trail. If the DBMS stores its own audit records (perhaps in the form of database tables to ease subsequent analysis), the DBMS TCB will have to directly provide the required access controls to the audit trail. This can be done using protection mechanisms already built into the TCB for MAC or DAC control. For example, in an MLS DBMS, the audit trail could be given a special compartment not included in the clearance of any regular user. Then no regular user could access the audit trail through standard untrusted applications. Only trusted applications with the required privilege could read or write the audit trail.

In general, it is dangerous to use standard DAC controls to protect the audit trail from modification. Any user or application which can run with the identity or privilege of the ‘‘owner’’ of the audit trail could then modify the audit trail.

4.5.3 System Availability vs. Audit Collection

Audit data can be indirectly deleted if the AIS is allowed to run while the storage media available for the audit trail is full. Either ‘‘old’’ audit records will be overwritten and lost or additional audit records cannot be written. The AIS will need to provide mechanisms to support the DBSSO in balancing the requirements for audit and system availability. This should include early warnings when space limitations are in danger of being reached, and the ability to archive older audit records while the system continues to capture new records. In fact, the TCSEC has been interpreted to ensure that audit data is not inadvertently lost due to the lack of available storage media [Interp 95]. Interpretation I-0005 was effective as of 1993-10-20 and applies to classes C2, B1, B2, B3, and A1:

> The following interprets the requirement that ‘‘The TCB shall be able to create, maintain and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects.’’
>
>> When the TCB becomes unable to collect audit data, it shall give a clear indication of this condition and take a pre-specified action. Tide system implementation may allow an administrator to choose from a range of actions. One of the actions that may be chosen by the administrator (or, if no choice is possible, the only action) shall be that the system cease performing auditable events when the TCB is unable to collect audit data. Choosing an audit overflow action shall be considered a security-relevant administrator event for the purpose of auditing. The TFM shall fully describe the administrator’s options.

To the extent the OS audit trail is used to hold DBMS audit records, the OS will provide the required mechanisms. It should be noted however, that the space limitations are likely to be reached more frequently with DBMS auditing because of the large quantity of data which will be captured when strong auditing is enabled. When audit data is captured in DBMS tables, the audit trail may be competing for space with user data. In this case, the DBSSO will need to set space limitations and keep track of space utilization to guarantee that there is sufficient space available for the audit data.

SECTION 5

AUDIT TRAIL ANALYSIS

The goal of an audit mechanism is to provide authorized system personnel with a means to regularly review a documented history of selected descriptions of activity on the system. If a system violation should occur, this documented history will log and permit reconstruction of the events leading up to and including this violation. The documented history also permits surveillance of users' activity. This may provide notice of successful or unsuccessful attempts to violate system security so that the unauthorized activity may be acted upon in advance of a potential later successful violation. The data must be easy to understand, easily retrievable, and easily analyzed. In addition, the tools for analysis should be both simple to use and integrated into the ordinary capabilities of the AIS.

5.1 REQUIREMENTS

In addition to the actual recording of all events that take place in the database, an audit trail should also provide high-level support for querying the audit data [Kogan 91]. The TCSEC accountability control objective states that an audit trail must have the capability ‘‘for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time and without undue difficulty” [DoD 85]. However, the TCSEC and TDI leave open the amount of audit trail processing required of an AIS (selective auditing is sufficient).

The increased sophistication of a DBMS is intended, in part, to provide the user with a greater degree of flexibility in posing a query. But this increased flexibility may open the system to additional security threats that can only be handled by auditing. This situation is particularly true for those DBMSs targeted at Class B3 or A1 which require audit mechanisms to be able to monitor the audit trail for imminent violations [DoD 85]. Aspects of this requirement have been interpreted.

Interpretation I-0084 was effective as of 1994-07-12 and applies to classes B3 and A1:

> The following interprets the requirement that ‘‘...and if the occurrence or accumulation of these security relevant events continues, the system shall take the least disruptive action to terminate the event.”
>
> > The action taken to terminate an imminent violation shall eliminate the capability to repeat the event, at least until a recurrence of the event would not indicate an imminent violation. The product developer shall provide a convincing argument for why the proposed action is the ‘‘least disruptive” of possible actions.

Interpretation I-0172 was effective as of 1994-04-19 and applies to classes B3 and A1:

> The following interprets the requirement that ‘‘The TCB shall contain a mechanism that is able to monitor the occurrence or accumulation of security auditable events that may indicate an imminent violation of security policy. This mechanism shall be able to immediately notify the security administrator when thresholds are exceeded...”
>
> > Audit of imminent violations is required only for actions that may lead to an actual violation (i.e., multiple failed login attempts, excessive use of identified auditable covert channels). The threshold for notification of the security administrator may be either settable by an administrator or fixed by the vendor. There shall be an immediate notification mechanism included with the product (e.g., a message to an identified terminal, a red light as part of the

> hardware), and the TFM shall describe how the security administrator can monitor this mechanism.

Ideally, a DBMS audit mechanism would have the ability to detect, deter, and alert the DBSSO of misuse patterns. However, automated misuse monitoring requires greater sophistication than is traditionally provided by a DBMS audit mechanism. The TDI adds that ‘‘if the total audit requirement is met by the use of more than one audit log, a method of correlation must be available” [TDI 91].

5.2 AUDIT PROCESSING

Audit review may be categorized by the timeliness in which it is completed. In the standard mainframe environment, manual audit review was said to have occurred after the DBSSO ensured that the audit mechanism appeared to still be working producing a stack of printout. Audit trails would be stored for a designated time and kept available for more detailed review subsequent to the identification of a compromise by some other means. The audit trails were thus used as a reference for damage assessment if needed and perhaps as evidence for pursuing a conviction. With the advent of simple audit analysis tools, audit reduction/formatting could be batched at the end of the week for after-the-fact weekend review or at night for morning review. The next advance in terms of timeliness permitted near real-time querying against a processed audit log. True real-time audit review increases the chances of catching an inappropriate activity in progress. Finally, support of early warning suspicion quotients and logging of DBSSO comments present the increased possibility of identifying likely future intrusion attempts [Halme 94].

Audit processing can range in order of sophistication [Arca 94]:

1. Audit Formatting,
2. Audit Reduction,
3. Audit Querying,
4. Static Profiling.

These audit analysis approaches are described in order of increasing sophistication.

5.2.1 Audit Formatting

Audit formatting is the straightforward formatting of logged data for manual review. The TCSEC has been interpreted to ensure the details of the audit record are documented [Interp 95]. Interpretation I-0046 was effective as of 1994-07-12 and applies to classes C2, B1, B2, B3, and A1:

> The following interprets the requirement that ‘‘The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of event shall be given.”
>
> > The documentation of the detailed audit record structure may describe either the raw records produced by the audit mechanism or the output of an audit post-processing tool as long as the records described contain the information specified in the audit requirement. If the output of the audit post-processing tool is described, the tool is considered part of the TCB.

5.2.2 Audit Reduction

Audit reduction is the simplest and most straightforward approach to developing an audit analysis facility which allows a DBSSO to better digest audit data. Innocuous audit data is eliminated and only the remaining filtered and formatted data is displayed for manual review. The innocuousness of audit data is determined based on a crude definition such as ignoring commands that do not modify certain secured data. Analysis of the remaining data is largely performed as before by manual review.

Lichtman and Kimmons developed an audit reduction paradigm to reduce the amount of data to be analyzed without degradation in the quality of the analysis [Lichtman 90]. They conclude that if redundant lower level events are removed in a consistent manner, the quality of the anomaly analysis might even improve while reducing the load on the analysis process/machine.

5.2.3 Audit Querying (Database Support)

A more flexible distillation of raw audit data, this approach supports the use of canned or on-the-fly querying of audit data after it has been populated into a database facility. With audit querying, a DBSSO can quickly determine irregular subject behavior. This approach does not maintain user or system profiles based on historical data, but may be manually configured to check activity against hard limit figures set by the DBSSO. Examples of audit querying include the models developed by Jajodia and Oracle which are described as follows.

Jajodia et al., have developed a model for databases that imposes a uniform logical structure upon the past, present, and future data, resulting in zero-information loss [Jajodia 89, 90; Kogan 91]. It provides facilities for both recording the history of updates and queries as well as querying this history. The goal of the model is to provide a convenient mechanism for both recording and querying accesses to a database in order to fully reconstruct the entire database activity.

The Oracle audit trail can optionally be stored in DBMS tables [Oracle 94a]. To read audit trail data, suitably privileged users can select from predefined database views of the audit trail stored in the database. Similarly, users can employ standard SQL or ordinary end user query tools to manipulate views of the audit trail, perform a variety of analysis queries, and produce reports which may be stored for later use.

5.2.4 Manually Trained Audit Analysis Tools (static profiling)

A more useful tool performs statistical analyses isolating obvious deviations from earlier learned thresholds for normal activity (e.g., flagging a user suddenly issuing a barrage of commands at a high rate per second who previously had been described to the tool as having an average command rate of one per minute). The DBSSO now needs to look only at a much reduced statistically deviant subset of the audit trail. User norms would be reevaluated only on some limited amount of “training” data and only when directed by a human during a learning phase. The evaluated DBMSs have not included static profiling tools.

5.3 INTRUSION DETECTION

Intrusion detection is one anti-intrusion method that is very dependent upon the quality and quantity of the audit data [Halme 95]. In general, raw audit data currently provides minimal real-time or near real-time benefit because the quantity and complexity of the data prohibits meaningful manual interpretation. Sophisticated intrusions and insider abuse may not even be discernible by single incriminating events and would escape even concentrated manual analysis. Typically, audit trails have been used far after the fact when an intrusion is suspected due to other reasons (e.g., a user reports a file has been maliciously modified, or a system operator happens to notice unusual modem activity). Only then are the audit trails carefully examined to assess damage and accumulate legal evidence. While not required by the TCSEC or the TDI, only with the aid of automated analysis tools would a DBSSO realistically be able to effectively detect a pattern of evidence identifying undesirable activity.

In fact, one of the key problems in using audit data to detect intrusions is that tremendous amounts of audit data must be collected and processed. The database audit tables may be flush with data which is irrelevant or may simply lack the data which would be useful and/or necessary. Even if the necessary data is successfully captured, it may be spread over time and across audit records, making it necessary to grasp the meaning of interrelationships between audited events. There must be a method of eliminating superfluous or misleading data to prevent excessive record keeping at the same time as retaining the essential data necessary as evidence in possible court cases [Hamilton 92]. This will determine clues such as whether the intruder knew what he was looking for. Intrusion countermeasures providing autonomous reaction to audited activity can react without a manual reviewer and at machine speeds against automated attacks. However, false alarms could be an inconvenience.

5.4 CORRELATION OF AUDIT TRAILS

It is important to collect and correlate audit data from a number of different abstractions to permit meaningful audit analysis [Hamilton 92]. This is particularly applicable if access mediation is performed by multiple mechanisms at different levels of abstraction [Schaefer 89b]. Since it can be difficult to trace transactions from events occurring at different layers [Schaen 91], the TDI requires that the audit trail generated by the DBMS include sufficient data to correlate DBMS events with OS events [TDI 85]. This includes ensuring that the clock times recorded in the audit trail are synchronized as well as ensuring that DBMS events can be associated with the corresponding OS user.

Audit analysis in a multiple TCB configuration can potentially be performed by the DBMS or by the OS. Since audit records for some database audit events (such as STARTUP and SHUTDOWN) must always be directed to the OS, correlation will have to be done between the two audit trails to produce a complete picture of a user's activity. The OS audit reduction facilities can be used for analysis of database audit records written to the OS audit trail. These facilities may have limited utility for examination of database records depending on the interface provided by the OS for audit record insertion by applications. However, the evaluation community has determined that manual review of the audit trails being correlated meets the intent of the TDI interpretation. Product vendors need to consider what data items (such as username and time of event) will be used for correlation and document this in the TFM. Alternatively, the DBMS can load the OS audit trail into a table. Once in a database, the OS audit records and the database audit records can be joined by use of a common identifier (e.g., OS username) using normal SQL processing [Oracle 94b].

At issue is the correlation of audit trails from various sources. An analysis tool may have a very hard time if the audit trail content from one system is radically different from another because the two systems are so different (e.g., if you compare the type of data audited by UNIX vs. what is audited by a relational DBMS vs. a network or object model DBMS).

5.5 DETECTION OF INFERENCE AND AGGREGATION

To detect inference and aggregation attacks (which is not a TCSEC or TDI requirement), it may be necessary to perform pattern recognition on a history of queries submitted by multiple users over multiple sessions [Schaefer 89b]. To detect such attacks, an audit subsystem capable of addressing this objective might need to include in the audit trail the raw text of user queries, compiled parse trees/execution plans, mediation decisions, and even retrieved data. The tools required to analyze such data would likely include an expert system and deductive inference engine capable of supporting complex pattern matching searches and analysis of the audit trail. The time required to accomplish this analysis could seriously impact performance if any sort of real time detection was desired. Even off-line analysis could be painfully slow [Hosmer 94].

5.6 DETECTING EXPLOITATION OF COVERT CHANNELS

The TCSEC requires that Class B2 and above systems audit all operations that may exercise covert storage channels [DoD 85]. Schaefer et al., suggest that to detect exploitation of covert storage channels, it may be necessary to audit hidden operations on objects internal to the DBMS [Schaefer 89b]. For example, in some DBMS architectures, it may be possible to manipulate the resource locks used to implement mutual exclusion for covert signaling. It is an inherent property of resource locks that their use may introduce covert channels into a system. This is particularly true in a distributed database. Two-phase commit protocols or other more complex synchronization and recovery protocols used in these configurations may have access delays that can be detected with a large number of locking events that can be manipulated. This suggests that an audit subsystem may need to record and correlate events from a much lower level of system abstraction than those directly available to a user in order to be able to detect exploitation of covert channels.

Covert channels can even be of interest to Class B1 DBMSs given the potential bandwidth of covert channels in an integrity enforcing MLS DBMS. Trusted Oracle provides an audit option that can be used for monitoring potential signaling channels [Oracle 94b]. This is the EXISTS audit option, which audits all SQL statements that return an error because the specified object, or a field in a tuple with an entity integrity constraint, or a value in the object limited by an integrity constraint, already exists.

SECTION 6

ARCHITECTURE

We have noted previously that many aspects of auditing in a DBMS are closely tied to the architecture of the DBMS. In particular, the architecture affects the assurance that one can place in the audit trail, and the volume and granularity of the audit data collected. To demonstrate this point, this section describes typical MLS DBMS architectures and notes the impact of each architecture on auditing. While a brief summary of each architecture is given, readers are assumed to be familiar with these architectures. For more details on the various architectures, the readers should refer to the appropriate literature.

6.1 INTEGRITY-LOCK

The Integrity-Lock architecture was one of the DBMS architectures considered in the 1982 Air Force Summer Study on Multilevel Data Management Security [Air Force 83]. The Integrity-Lock architecture was one of the approaches considered appropriate for achieving some form of multilevel security in the near term (five years or less). Currently, only one vendor, Atlantic Research Corporation, has developed a product based on this architecture [Knode 89]. For more data on the Integrity-Lock architecture, see [Graubart 84a, 84b].

6.1.1 Architectural Summary

The basic Integrity-Lock architecture consists of three components as depicted in Figure 6.1: an untrusted front-end process, an untrusted back-end process, and a trusted filter. The trusted filter is used to mediate data and queries between the untrusted front end and the untrusted back end. Upon data insertion, the filter associates a label and a cryptographic checksum with the data. The data, label, and checksum are then passed to the untrusted back end, which inserts all of the data into the database. Because the data is stored in the database in unencrypted form, the untrusted back end can examine the data as it would any other data. When a query is issued by the user, the back end performs logical qualification DBMS functions on the data to locate and retrieve those records that satisfy the query. The retrieved data is then passed through the filter. The filter performs two checks. First, it determines whether the label associated with the retrieved data is dominated by the security level of the user. If it is not, then the data is not returned to the user. Next, the filter verifies the validity of the checksum, and a correct checksum indicates that neither the data nor the label have been altered. If both of these checks are satisfied, then the data is passed to an untrusted front-end process that is running at the same level as the user. This process then performs any necessary formatting before passing the data to the user.

In some variations of the Integrity-Lock architecture, there is a fourth component—an untrusted parser that runs at the same security level as the user [Graubart 84b]. The parser is responsible for parsing the user query and generating an appropriate parse stream which it provides to the untrusted front and back ends. In those systems where the parser is not explicitly called out, its function is handled by the untrusted front end.

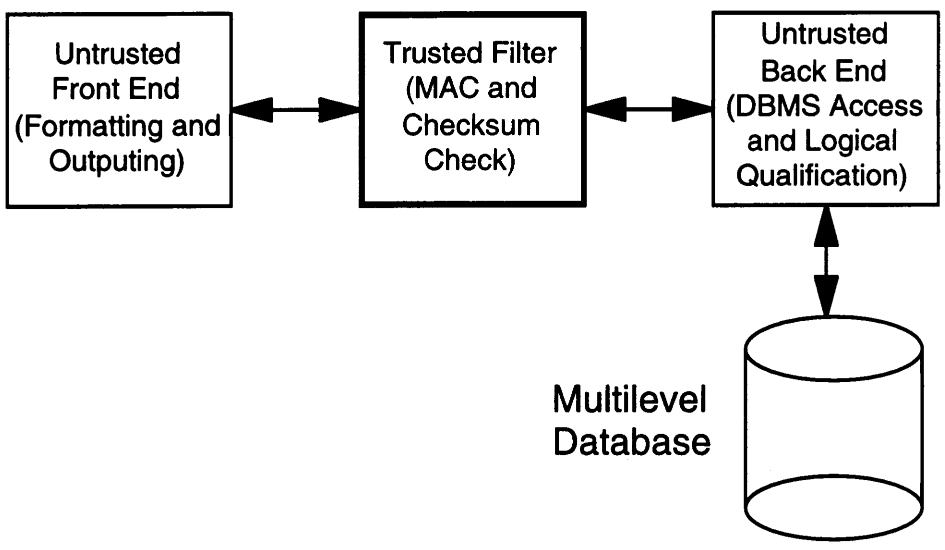

Untrusted Front End (Formatting and Outputing)
Trusted Filter (MAC and Checksum Check)
Untrusted Back End (DBMS Access and Logical Qualification)
Multilevel Database

Figure 6.1: Integrity Lock Architecture

Schaefer pointed out in the 1979 Air Force Summer Study at Draper Labs that the Integrity-Lock architecture possesses a major security flaw; the untrusted DBMS has complete access to all of the data in the database [DeWolfe 79]. Therefore, the DBMS could pass back unclassified information to a user, but organize it in such a manner that it was actually signaling back classified information (for example, the first letter of each tuple returned could spell out some classified information). While there are ways to limit this channel, it is not possible to close this channel. This channel can be exploited by a Trojan horse in the DBMS process and some means by which a user could signal this Trojan horse. In addition, the user may be able to pose a query in which the logical qualification will be based in part on classified records in the database and as a result yield a set of releasable records which depends on classified data in the database.

Despite the security flaw already mentioned, there is some appeal to the Integrity-Lock architecture. In particular, it can be constructed out of existing DBMSs, and little if any change to the DBMS code is required. It may be useful in environments with a sufficiently benign threat environment or where only a small number of very large records is ever retrieved.

6.1.2 Auditing Issues in the Integrity-Lock Architecture

Because the back end of the Integrity-Lock is not part of the TCB, it cannot be trusted to provide correct audit data [Schaefer 89b]. The only component of the architecture that can be trusted is the trusted filter. Thus, at a minimum, the filter should be able to audit all of the data it passes to the untrusted front-end process.

Given the nature of the threats to which the Integrity-Lock is subject, additional auditing may be required. All of the data passed to the filter from the back end should also be auditable. This information may help to detect indications of when an attempt to violate security is occurring. It is advisable that the query and the parse stream generated as a result of the query should also be auditable.[3] The inclusion of the query and the parse, while not helpful in real time, might be of use in retrospective analysis of the audit log. Well-designed audit reduction tools could check the consistency between the query and the parse stream. Any inconsistencies might indicate a possible Trojan horse in the parser attempting to signal the back end.

Capturing all of this information only provides hints of the actual actions of the back end. Because the back end is untrusted, it is not possible to reliably detect malicious activity of the back end. Not only is it not possible to audit malicious actions of commission by the back end, but it is not possible to audit malicious actions of omission. For example, assume that a tuple needs to be upgraded. The actual upgrade occurs at the filter. It is the responsibility of the back end to insert the upgraded data into the database replacing the original tuple. Because the back end is not trusted, the action cannot be assured to have taken place, nor can any audit of the action be trusted.

Note that if any DAC or database integrity capabilities are provided, this will be done in the back end also. Thus, there is no way to audit whether these actions were conducted.

In terms of volume, the amount of audit data generated is less than in other DBMS architectures. The back end only passes tuples to the front-end that satisfy the logical qualification clauses. Except for queries of the nature ‘‘Select * ...” the number of tuples that are likely to be passed to the filter is likely to be reasonable in number, and so will be the volume of audit data generated.

6.2 DISTRIBUTED DBMS ARCHITECTURE

Like the Integrity-Lock architecture, the Distributed DBMS architecture was also discussed at the Air Force Summer Study on Multilevel Data Management Security [Air Force 83]. Prototypes based on the architecture have been developed [O’Connor 89; Kang 94].

6.2.1 Architectural Summary

In the architecture depicted in Figure 6.2 there is a single trusted front-end node [Kang 95]. This node interfaces with multiple back-end nodes, each of which is untrusted. It should be noted that the issue discussed here would be the same whether the nodes were actually implemented on different machines or on a single machine using an OS TCB to control access to the nodes. The back-end nodes each run at a different security level. The data in the lower level nodes is replicated in the higher level nodes. Thus, the Unclassified node just has Unclassified data, the Confidential node has Confidential and Unclassified data, the Secret node has Secret data, Confidential, and Unclassified data, and so on.

When a user issues a query, the trusted front-end node determines the level of the user and issues the query to a node at the corresponding level. The back-end node performs all of the necessary DBMS functions (e.g., selection, project). The extracted data is then passed back to the front-end node, which labels the data (reflecting the level of the node from which it was extracted) and passes it to the user. In the case of an update, the front-end node is responsible for ensuring that the update is replicated up to the higher level back-end nodes.

3.We are not suggesting that the parser be included in the TCB, but that a copy of both the query and parse stream be passed to the TCB. The ease or difficulty of doing this varies with the specifics of each Integrity-Lock implementation.

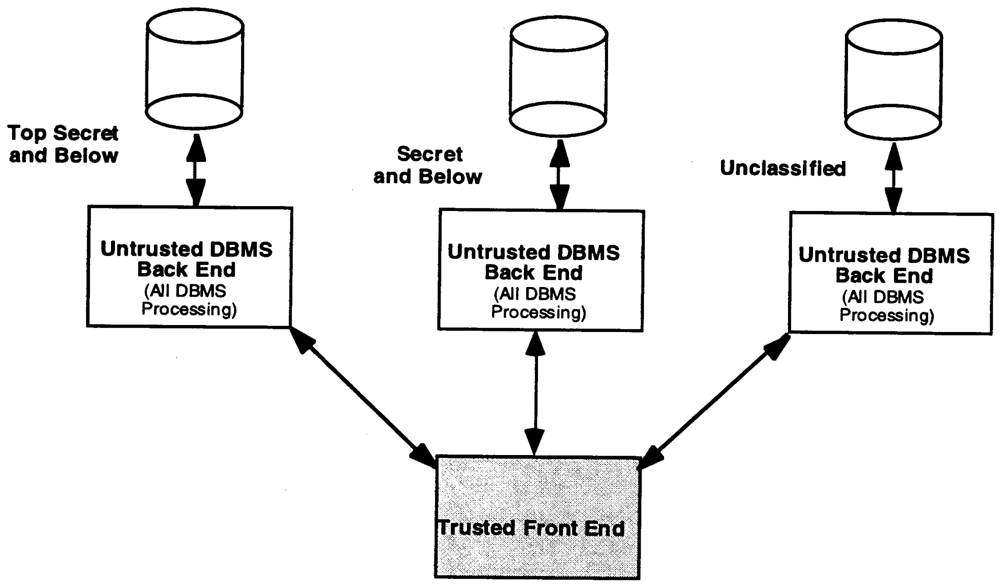

Top Secret and Below
Secret and Below
Unclassified
Untrusted DBMS Back End (All DBMS Processing)
Untrusted DBMS Back End (All DBMS Processing)
Untrusted DBMS Back End (All DBMS Processing)
Trusted Front End

Figure 6.2: Secure Distributed DBMS Architecture

The advantage of this architecture is that, like the Integrity-Lock architecture, it can be built out of untrusted DBMSs. Unlike the Integrity-Lock architecture, it does not suffer from a major security flaw. The one significant weakness of this architecture is that it requires one node per security level. In an environment with a large number of levels and/or compartments, this can be a major problem because of the large number of single level databases required.

6.22 Auditing Issues in the Distributed DBMS Architecture

As with the Integrity-Lock Architecture, in the distributed DBMS (DDBMS) architecture the back ends are also untrusted. This means that any auditing provided by the back ends would be suspect [Schaefer 89]. Only the front-end node is capable of providing a trustworthy audit trail. produced by the front end consist primarily of the data returned by the selected back-end machine. As with the Integrity-Lock architecture, this information is only partially. To make the data more useful, this audit mechanism in the front end should be capable of capturing the user query.

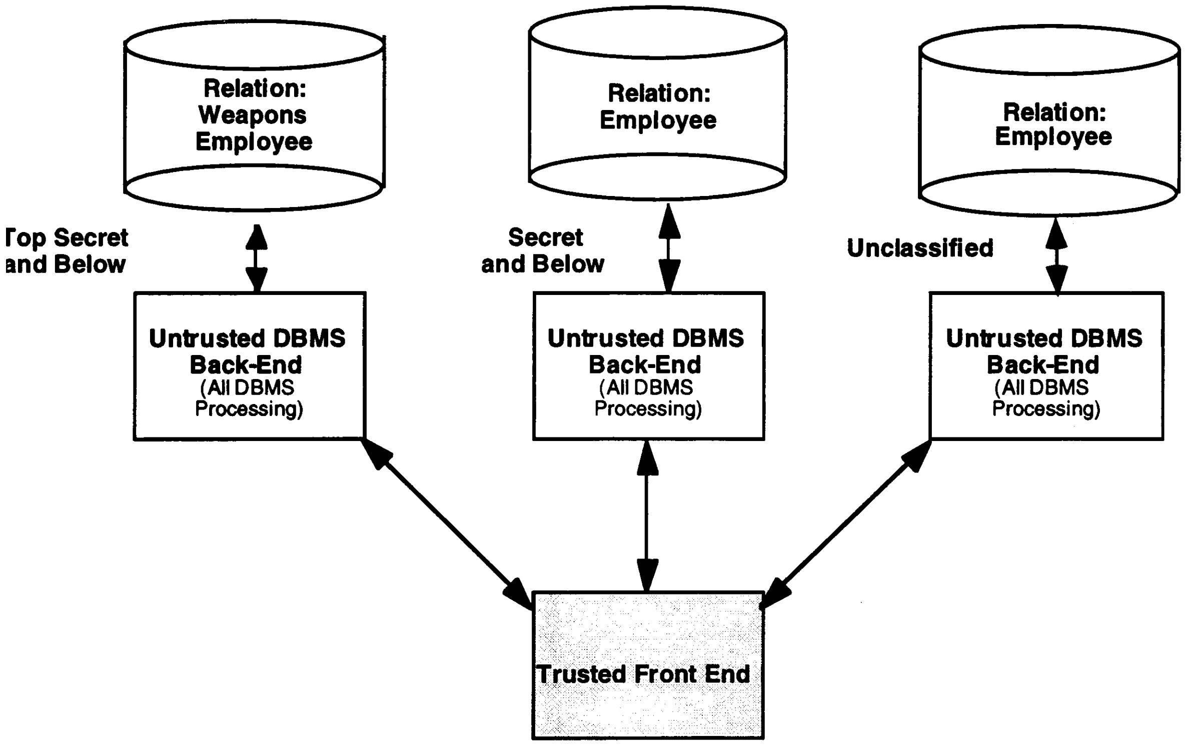
In the pure sense of this architecture with totally back ends, no attempted DAC or integrity policy violations could be audited because these policies, if enforced, would be handled entirely at the back end and any auditing of these policies, like any enforcement of the policies, would be untrustworthy. Thus, only MAC-related objects (storage) and MAC accesses and policy violations can be audited under this architecture. If MAC enforcement is achieved with the DDBMS architecture, but the back ends are trusted to enforce DAC or integrity policies, attempted violations of these policies could be audited with whatever degree of assurance has been established for the back ends.

The MAC objects audited would essentially be the data returned by the back ends (the label associated with this data would be the label of the back-end machine). A MAC access would be any access or requested access to the back-end machine. Given the nature of this architecture, the attempted MAC violations that could occur are queries targeting a relation that only exists in higher level back-end machines (see example in Figure 6.3).

Of course, the attempted violation would be prevented from actually taking place because the front-end does not permit any data to be returned from a back-end unless the level of the back-end is dominated by the level of the subject making the query. However, in order to detect and audit such an attempted access, the front-end would need access to the names of relations, attributes, and data structures that exist at different levels. Since they are maintained by untrusted DBMSs, the front-end cannot know this data reliably. Alternatively, the audit mechanism could simply record all instances where the queried back-end DBMS indicated the referenced structure did not exist. However, the reliability of this data is still suspect since it is generated by an untrusted DBMS.

The nature of this architecture also precludes covert channels if the nodes are in fact separate machines. If they are running on a single machine under the control of a TCB, they can potentially covert channels permitted by the OS TCB. However, they do not introduce additional covert channels. As such, the requirement to audit covert channels is not applicable.

Depending upon the specifics of the MAC policy, there is one aspect of MAC policy that might not be auditable under this architecture. If the MAC policy includes the ability to regrade data, then this action cannot be audited under this architecture. This is because of the unique way in which regrades are handled. For example, to upgrade a tuple under this architecture requires the deletion of the tuple from the lower classified machines in which it exists. This action upgrades the data since the lower level data is replicated in the higher level machines. Since the back-end machines are all untrusted, it is not possible to audit the deletion of the tuples (even worse from a MAC policy perspective, one cannot even ensure that the deletion takes place, thus precluding trustworthy enforcement of any upgrade policy)


Relation: Weapons Employee
Relation: Employee
Relation: Employee
Top Secret and Below
Secret and Below
Unclassified
Untrusted DBMS Back-End (All DBMS Processing)
Untrusted DBMS Back-End (All DBMS Processing)
Untrusted DBMS Back-End (All DBMS Processing)
Trusted Front End

Request from unclassified user: Select * from Weapons

Figure 6.3: Example of Unauthorized Access Request

Thus far, the DDBMS architecture has been discussed with the assumption that the back ends are untrusted, since this is the basic architecture. But in recent years, there have been some proposed variations to this architecture in which the back ends are trusted, but not to the same degree as the front end. For example, in some cases the back ends are C2 nodes with C2 DBMSs, and the front end is B1 or higher. This variation in the architecture provides for additional auditing. The actions of the back ends with regard to DAC and integrity policies then in force can now be audited by the back ends. In theory, these audit trails can now be combined with the audit trail generated by the central node to give a more complete audit picture. In effect, the back ends would audit the non-MAC policies and the front end would audit the MAC policy.

The degree of assurance one can place in the accuracy of the audit records written by the back ends is limited by the lower assurance given the back-end DBMSs. There are two perspectives of this question. The first is that C2 assurance is sufficient for the mechanism controlling and auditing the functionality required in C2 systems. That is, C2 assurance would suffice for audit of DAC. Under this perspective, there is no problem with combining the two audit trails. However, a C2 system and its associated audit mechanisms are more vulnerable to compromise than a higher assurance system. As such, the audit trails generated by the lower assurance back ends cannot be trusted to the same degree as the higher level assurance front-end node. Hence, some would not consider audit-data gathered by a mechanism of lower assurance to be acceptable as part of a system with higher assurance requirements. Essentially, this is the question of balanced versus uniform assurance but applied to auditing. There is not a general consensus on this issue.

At first it might appear that the auditing capabilities of the C2 devices might help alleviate some of the MAC auditing concerns raised earlier, in particular the problem of auditing regrading of objects. But that is not true. While the C2 back ends may be trusted to delete database objects, the level of trust in the back ends is less than in the front end. As such, the lower assurance back-end devices are being used to support a higher assurance MAC policy and the auditing of that policy. Such an arrangement is suspect. In addition, the DAC objects in most DBMSs are relations, views, and attributes. While it is not unheard of for a DAC object to include tuples, it is not common either. Thus, there is a possibility that what constitutes a MAC object (e.g., tuples) is not an object from the perspective of the back end but rather, data within an object. If that is the case, then the back end may be unable to audit the insertion or deletion of the tuples.

6.3 TRUSTED SUBJECT

The trusted subject architecture, also sometimes called a *dual kernel-based architecture* [Graubart 89], was one of the later MLS DBMS architectures to be developed. It was not part of the Air Force Summer Study. Rather, it grew out of the work of various vendor efforts and prototype efforts. For detailed descriptions of some of the trusted subject architectures, see [Winkler-Parenty 90].

6.3.1 Architectural Summary

This approach has the DBMS perform labeling and also access mediation for objects under its control. MAC controls on DBMS objects are enforced by the MLS DBMS. The architecture is referred to as a trusted subject approach because the DBMS has the privilege to override the MAC policy enforced by the underlying OS TCB. It must be trusted not to misuse that privilege. Hence, it is considered a trusted subject of the underlying OS TCB. Under the trusted subject approach, a single DBMS has access to all of the data in the database. The architecture depends upon the underlying trusted OS to protect and isolate the DBMS and the associated database(s). This is generally handled by having the DBMS and the database maintained at system high, sometimes along with a special compartment.

The advantage of this architecture are that it can provide good security, and its performance and data storage overhead are relatively independent of the number of security levels involved. Its disadvantage is that the DBMS code that performs access mediation must be trusted, and often such code is both large and complex. Thus, a significant amount of trusted code may be required under this approach [Hinke 89]. Despite this disadvantage, this architecture is the single most popular approach. Some variation of this approach is employed by Oracle, INFORMIX, Sybase and Ingres (Computer Association) in their MLS DBMS products.

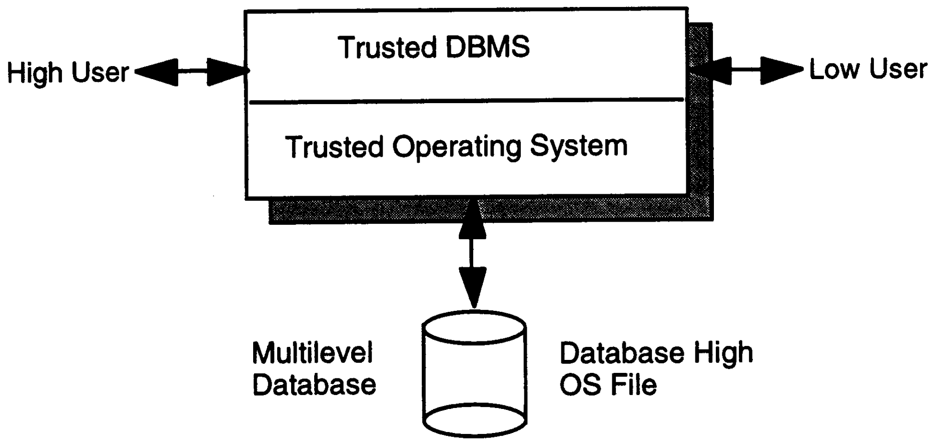


Figure 6.4: Trusted Subject Architecture

6.3.2 Auditing Issues in a Trusted Subject Architecture

As noted above, in this architecture, the MLS DBMS handles all accesses to the database, and so it is different than the other architectures discussed so far where untrusted code performed the database accesses. It is the first true MLS DBMS that has been discussed. But there are multiple variations possible in the trusted subject architecture. One aspect contributing to the variation is how much of the DBMS should be included in the TCB, in particular, the non-security-relevant functionality. Another aspect pertains to the ordering of certain DBMS and security functionality. Both of these issues will be examined as they pertain to auditing.

Let us return to a question raised in Section 3.3: what accesses need to be audited in a DBMS and, in particular, in the trusted subject architecture? One possibility is to audit only the data returned to the user. If this is done, the ability to audit unauthorized access attempts will be lost. Alternatively, the data could be audited as it is accessed by the DBMS. But this action could result in an audit of every access that the DBMS performs while sequentially working through a relation, trying to determine which tuples satisfy the logical qualification clause of a query.

Ideally, there are three points in a trusted subject MLS DBMS where auditing might be performed. The first is where the data is returned from the TCB to the user. This is a logical point to audit because it is consistent with the basic definition of auditing in the TCSEC. That is, it is consistent with the data that is passed into the user's address space. But auditing at this location does not provide insight into any unauthorized access attempts performed by the user.

These attempted access violations can only be detected earlier, as the data is retrieved by the DBMS. Therefore, the second point is when the DBMS performs the FETCH operation from the database. As noted earlier, the audit of every access that the DBMS performs while sequentially working through a relation trying to determine which tuples satisfy the logical qualification clause of a query is undesirable. How then should auditing of database accesses be handled? Auditing at this point is desirable because any data that passed the logical qualification process would presumably be data that the user was attempting to access, and thus data that was rejected for security reasons and may be suspect. In some queries, a tremendous amount of data will be rejected for security reasons (an example is a query Select * from EMPLOYEES). In such a case, the rejected data may not be indicative of an intended security violation, but more indicative of the user's ignorance about the existence of the data. However, m general, rejected data can indicate an attempted security violation. The only way to determine if the violations were intended is to audit the user's query and correlate tuples rejected for security reasons with the query that was issued. This brings us to the third point in a trusted subject DBMS where auditing is required - at the beginning to capture the user query. The reasoning behind including the query in the audit trail was discussed in previous sections.

Thus, it is desirable to be able to audit data returned to the user, the query itself, and the data that passes logical qualification but fails security filtering. It should be noted that if the parser is not trusted, there is no assurance that the query which is audited is the one which is actually executed. Also, auditing data which passes the logical qualification but fails security filtering only makes sense if the logical qualification is performed before the security filtering as depicted in Figure 6.5.

Unfortunately, while this approach may be desirable from a security audit perspective, it is less than ideal from a security architecture perspective. Let us first look at the query parser. Making the parser trusted increases the size and complexity of the TCB. At lower levels of assurance, this is not an issue. But at Class B2 there is a modularity requirement imposed upon the security architecture. At Class B3 and higher, there is a minimization requirement imposed upon the TCB that effectively mandates the exclusion of non-security-relevant code from the TCB. Traditional or legacy parsers are large, complex, and generally ill-structured entities, the bulk of whose functions are not security relevant. Thus, the incorporation of such parsers into the TCB for the purpose of addressing the auditing security-concern described above would likely result in a system that was unable to satisfy the TCSEC architectural requirements.

A similar issue arises in regards to the ordering of the logical qualification processing and security filtering. The logical qualification processing is performed by a large and fairly complex set of code. In addition, logical qualification processing is generally not considered security relevant. As such, the logical qualification function should not be included in the TCB, so that a well-modularized and minimized TCB is provided. Providing such a minimum TCB, while still maintaining the basic concepts of the trusted subject architecture, requires that the security filtering precede the logical qualification processing as depicted in Figure 6.6. This is necessary to ensure that only trusted code handles the retrieved data prior to security filtering. By reversing the order of the security filtering and logical qualification processing functions, the ability to detect unknown access attempts is eliminated.

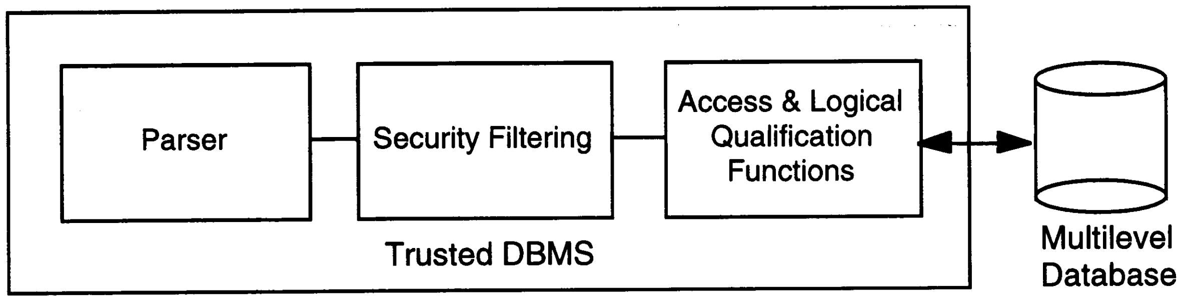


Figure 6.5: Trusted Subject Architecture Optimized for Auditing

Given current technology, the higher level audit requirements (monitoring the accumulation of security auditable events) and the higher level architecture requirements (modularity and minimization of the TCB) come into conflict regarding the inclusion of the parser and the qualification processing into the TCB. At this point, there does not seem to be a good solution to this conflict. The solution probably lies with obtaining a better understanding of what is meant by minimization in a DBMS, and with the development of a restructured parser qualification processor that would satisfy the B2 and higher architectural requirements, while still providing the necessary functionality.

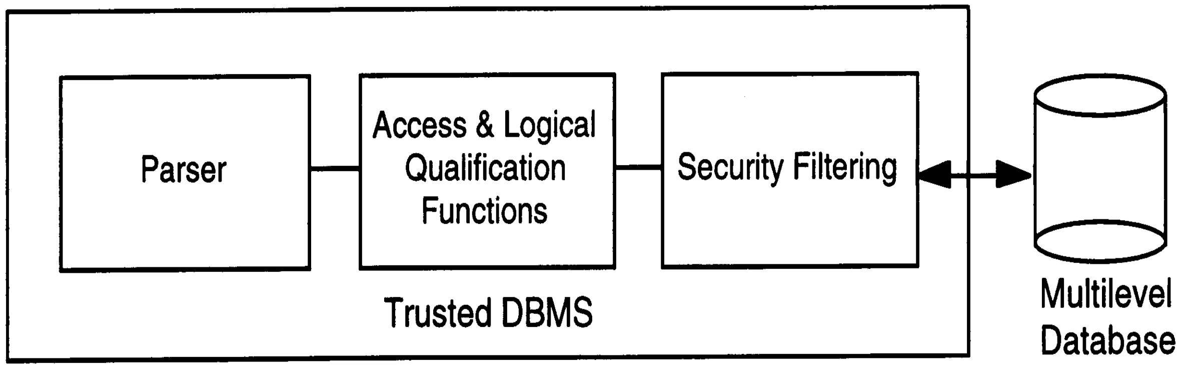


Figure 6.6: Trusted Subject Architecture Optimized for Modularity and Minimization

6.4 NO MAC FRIVILEGES ARCHITECTURE

The No MAC Privileges (NMP) architecture was discussed at the Air Force Summer Study on Multilevel Data Management Security [Air Force 83][4]. The SeaView research program has developed a prototype system based on this architecture [Hsieh 93]. Trusted Oracle can be configured to run in either this architecture (which they call OS MAC mode) or the trusted subject architecture (which they call DBMS MAC mode) [Oracle 94b]

4.We use the term 'No MAC Privileges' to describe this architecture, which has been variously referred to as TCB Subsets (TDI 91], Kernelized Architecture [Air Force 83], and OS MAC Mode (Oracle 94b]. These terms are all equivalent except that TCB Subsets is even more general, permitting the layering of any policies, rather than just DAC on top of MAC.

6.4.1 Architectural Summary

The NMP architecture relies on a decomposition of the multilevel database into multiple single level relations. The OS TCB provides MAC protection by controlling access to the OS objects in which the single level relations are stored. Each interfaces with an instance of the DBMS running at the user's working level. The OS MAC enforcement guarantees that the DBMS instance at a given level can only read those OS objects at levels dominated by the level of the DBMS instance. The DBMS instance can only write to OS objects at the same level as the DBMS instance. The DBMS is not given the privilege to override the OS MAC controls. Hence, it is not a trusted subject with respect to MAC. Figure 6.7 illustrates the NMP architecture. The description of the DBMS as untrusted refers to the fact that it is not relied upon to enforce the MAC policy. DAC enforcement is done by the DBMS. Hence, the DBMS is trusted to properly enforce DAC controls.

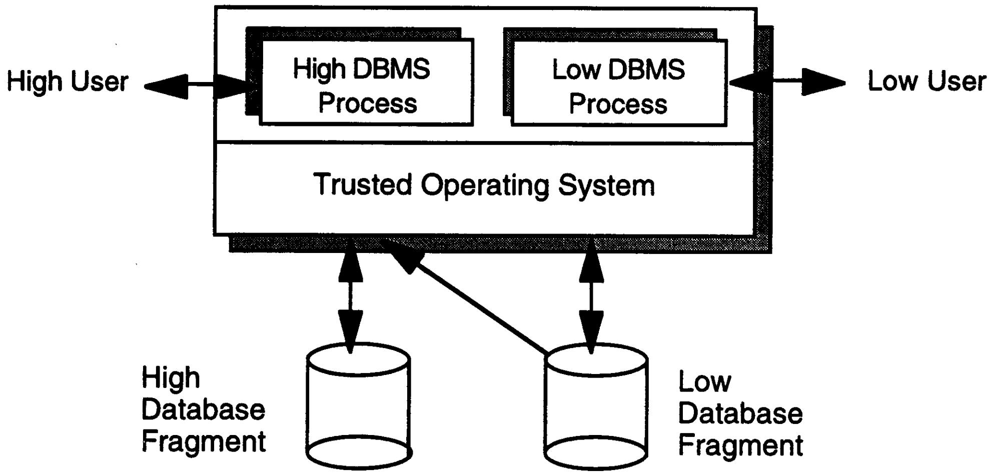


Figure 6.7: NMP Architecture

The NMP architecture has the advantage that the amount of code which is trusted to do MAC enforcement is significantly less than in the trusted subject architecture. The major disadvantages are the performance and storage implications of decomposing the multilevel database into multiple OS objects. This can represent a heavy penalty if there are many levels and compartments of data.

6.4.2 Auditing Issues in a NMP Architecture

The issues associated with auditing in a NMP architecture are similar to those for the DDBMS Architecture. The MAC enforcement is done by the OS TCB at the granularity of OS objects. Since there is no additional MAC enforcement done by the DBMS, there is nothing meaningful which it can add with respect to auditing for MAC. A DBMS instance at a particular level cannot observe database objects at any levels except those lower than or equal to the level of the DBMS instance. DBMS objects at higher or incomparable levels are stored in OS objects which cannot be accessed by the DBMS instance. Hence, the DBMS instance can neither be the instrument for revealing data to the user contrary to the MAC policy nor can it detect any attempts to violate that policy. It is inherent in the nature of this architecture that the security filtering is performed (by the OS TCB) before logical qualification takes place. Access control decisions are specifically required to be audited by the TDI [TDI 91] and bring up some difficult challenges to effective auditing.

On the other hand, the DBMS is trusted to enforce the DAC policy and it can therefore audit actions relevant to DAC. Indeed, one approach to implementing a NMP architecture is to use an existing

C2 DBMS to provide the DAC enforcement and associated auditing. The fact that one only has C2 level assurance in the correctness of the DAC enforcement and auditing mechanism raises the same issues as discussed in the DDBMS section above.

A NMP architecture also introduces several issues with audit trail correlation that were discussed in Section 5.4. In this configuration, one database audit trail exists at each level. While having multiple audit trails is effective in ensuring that lower level users cannot detect higher level activity, it can make tracing a pattern of suspicious behavior across the entire system difficult. In order to see a complete picture of the audited activity, the DBSSO can create a view of all lower level database audit trails in the highest level database. To determine the label of an audit record within this view, the user can select the Row_Label column from that tuple. Because setting the audit options is a write operation to the data dictionary, a user can set audit options only in the current database. Therefore, the TFM needs to point out the importance of having identical audit options set across all levels of the database [Oracle 94b].

6.5 TCB SUBSETS

A TCB consists of one or more sub-elements that together enforce a security policy over an AIS. Complex AISs distribute policy enforcement to various sub-elements. While determining the trust characteristics of a complex AIS on the basis of a collection of subparts is not well understood, there are approaches for arguing about composition of AISs.

The approach used in the TDI is to view the AIS as being composed of hierarchically-ordered systems. There is a ‘‘privilege” hierarchy characterized by dependency. The construct developed for dealing with layered systems is TCB subsets. If a TCB subset can be shown to be constrained and unprivileged relative to the more primitive TCB subset from which it obtains resources, then the scope of the assessment can be limited. With this concept, the TDI then defines the conditions under which the evaluation of a system can proceed with the maximum degree of independence of the evaluation of its individual TCB subsets. This type of evaluation is called evaluation by parts and the six required conditions for a system to be suitable for evaluation by parts are:

- The candidate TCB subsets are identified;
- The system policy is allocated to the candidate TCB subsets;
- Each candidate TCB subset includes all the trusted subjects with respect to its policies;
- The TCB subset structure is explicitly described;
- Each TCB subset occupies distinct subset-domains; and
- The more primitive TCB subsets provide support for the Reference Validation Mechanism arguments for less primitive TCB subsets.

The Trusted Subject Architecture cannot be decomposed into TCB subsets meeting these conditions because the DBMS is trusted with respect to the OS MAC policy. The NMP architecture on the other hand can be decomposed into an OS TCB subset and a DBMS TCB subset suitable for evaluation by parts.

For systems suitable for evaluation by parts, the TDI provides interpretation of TCSEC audit requirements. The interpretations require that the entire TCB must meet the TCSEC audit requirement through the cooperative action of the TCB subsets. TCB subsets may maintain their own audit logs or use an audit interface provided by a more primitive TCB subset. If a TCB subset uses different user identifications than a more primitive TCB subset, there must be a means to associate audit records generated by the actions of a given individual. This association can be done either when the records are generated or later. Any TCB subset in which events may occur that require notification of the DBSSO shall be able to:

1. Detect the events,
2. Initiate the recording of the audit trail entry, and
3. initiate the notification of the DBSSO.

The ability to terminate the recording and notification described in 2 and 3 above can be provided either in the TCB within which they occur, or in the TCB subset(s) where the event was initiated.

The TDI also points out that the monitoring and notification requirements may require cooperation among multiple TCB subsets or multiple instantiations of the same TCB subset. For example, detection of the accumulation of events for a single user may require collecting events detected in different TCB subsets.

6.6 SUMMARY

As can be seen from the above discussion of MLS DBMS architectures, the choice of a security architecture has a definite impact on the ability of the system to perform auditing. It is also apparent that there is a potential conflict between the requirements of a security architecture and those of security auditing. Further thought needs to be given to ways to address the inconsistency between these two requirements. Finally, it should be apparent from the discussion of the Integrity-Lock architecture that those architectures that are inherently flawed cannot correct or compensate for the flaws by auditing.

SECTION 7

DATA RECORDING FLEXIBILITY

This section presents a recommendation that has not been discussed in the literature previously nor is it required by the TCSEC or TDI but is a relevant contribution to the DBMS auditing problem. No current DBMSs provide this level of flexibility but it could be very helpful to some users if they were to do so. This recommendation addresses the ability to precisely adjust the amount of data that is recorded for particular audit events as follows:

Just as it is important for audit events to be auditable (meaning you can turn particular events on or off) it would also be helpful for the data captured to be *recordable* (meaning that you can select what specific data is recorded for that event, rather than recording all data that is available related to a particular event). This would allow a DBSSO to precisely configure the audit mechanism to best meet the (typically conflicting) needs of tracking what is happening without overflowing the audit logs or adversely impacting system performance. Most current trusted systems record a fixed set of data per audit event type rather than providing the flexibility to select or prefilter what data is recorded for each audit event. As described in Section 3, what they do provide is the ability to decide whether or not an audit record is generated at all (i.e., audit events are 'auditable').

For each auditable event, it would be beneficial to be able to select what data is recorded based on the same type of factors used to decide whether or not to audit an event in the first place, such as [Oracle 94a]:

- Object accessed
- User
- Operation performed
- Privileges used
- The sensitivity of the object accessed (for B1 and higher systems)
- Granularity of object accessed (e.g., table, tuple, element)

This would allow certain critical objects, certain users, certain operations, or use of certain privileges to be audited to a finer granularity (i.e., more data is recorded) while other less sensitive or interesting events would be audited at a coarser level (i.e., less data is recorded), or not at all. In fact, the ability to combine these factors could also be beneficial. For example, it might be useful to be able to record more detailed data whenever a particular set of users access a particular object, or whenever an access to that object occurs using a particular operation (e.g., update, or delete). The object granularity factor is particularly important because it can play a significant factor in the volume of audit data collected (e.g., is one record generated whenever a command accesses a table or is one record generated for every tuple accessed in the table). As described in Section 3, Trusted Oracle allows the DBSSO to turn auditing on or off based on these factors, but is does not allow the DBSSO to determine what data is recorded based on these factors. This means that if an audit record is generated, the contents of the record are fixed as predetermined by the developers of Oracle.

This flexibility to decide what data to record would enable the DBSSO to reduce the size of the audit trail that was generated but may affect the performance of the DBMS. Depending on the implementation of the filtering mechanism and the actual settings that are used, the DBMS could be faster or slower than a mechanism which does not support *recordability* or prefiltering. There is going to be some overhead involved in checking the filters associated with each event to determine what data to record. However, if a significant amount of data is filtered out, this could cause an overall speed up in the audit mechanism because it is not recording as much data, in addition to reducing the size of the audit trail.

SECTION 8

SUMMARY

As was described, there are a large number of issues involving auditing in secure DBMSs, many of which are unresolved, and are also dependent on the specifics of the architecture and implementation. This document explores these issues and describes known alternatives along with the pros and cons of each. There are, however, some specific recommendations that were presented. These are summarized here.

With respect to what information should be recorded, in addition to what is minimally required by the TCSEC, we recommended the following information be captured in the audit trail:

- Reason why each event failed, including the user's security levels or authorizations, or a pointer to this data.
- The original query, prior to parsing or optimization.
- Each audit event record should be traceable back to the query which caused it.
- Each query that is part of a transaction should be traceable to the transaction.
- Whether a transaction eventually was committed or was rolled back.

This information is very helpful in understanding a user's intent, and provides a better picture of what has actually occurred in the database.

A number of audit storage alternatives were described. Each of these alternatives has certain benefits and drawbacks. To provide flexibility, a DBMS should support more than one of these choices. For example, providing the ability to record a DBMS specific audit trail, merging the audit data with that of the host OS, or forwarding it to a specific location are all reasonable choices which should be supported. The actual method used should then be left to the DBSSO.

One of the significant problems with auditing today is poor support for audit analysis. DBMSs have an advantage in that by their very nature, they can be used to support audit analysis of their own audit trails if the data is stored in a database. Once stored in a database, the full power of the DBMS can be brought to bear to search the data for whatever is desired. This capability can be a powerful aide to a DBSSO searching for an intrusion and it can also be advantageous to the DBMS vendor since they do not have to construct an audit analysis tool from scratch in order to provide an audit analysis capability. Both INFORMIX and Oracle have the ability to read the database audit trail stored in an OS file into a database and then perform searches on that data using SQL. Oracle can also search the audit trail that is recorded directly into an audit database.

APPENDIX

AUDIT EVENT EXAMPLE

As an example, the events audited for INFORMIX-OnLine/Secure are provided in Table A.1 [INFORMIX 94].

Actions	Event Types	Data Collected
User	Session Startup	None
	Object creation/deletion/access	Object name, object label, object ID, label at which object was opened
	Database privilege granting/ revoking	Database name, grantor, grantee, privilege, revoke
	Table privilege granting/ revoking	Table name, grantor, grantee, privilege, revokee
	Transaction management	None
Administer	Chunk Management	dbspace number, chunk number, mirror status
	dbspace management	dbspace name, mirror status
	BLOspace management	BLOspace name, mirror status
	Transaction log management	None
	Read/delete audit masks	Audit mask name
	Create/Update audit masks	Audit mask name, audit mask
	Grant DAC privilege	Object name, object label, grantor, grantee, privilege
	Revoke DAC privilege	Object name, object label, revoker, revokee, privilege
	Access a database	Database name, label
	Database label modification	Database name, old label, new label
	Table access	Database name, table id, table label
	Table label modification	Database name, table name, old label, new label, owner
	Use of transient process	Command line used when process was executed

Table A1: INFORMIX-OnLine Secure Audit Event Types

REFERENCES

[Air Force 83]	Air Force Studies Board, *Multilevel Data Management Security,* National Research Council, National Academy Press, Washington, DC, 1983.
[Albert 92]	Albert, S., Ashby, V. A., Hicks, S. E., ‘‘Reference Model for Data Management Security and Privacy,” *ACCM SIGSAC,* Vol. 10, No. 2 & 3, Spring-Summer 1992.
[Arca 94]	Arca Systems, Inc., *Survey of Available Intrusion Detection System (IDS) Techniques,* ATR 94028, 18 November 1994.
[Audit 88]	*A Guide to Understanding Audit in Trusted Systems,* NCSC-TG-001, Version-2, National Computer Security Center, June 1988.
[Biba 77]	Biba, K. J., *Integrity Considerations for Secure Computer Systems,* ESD-TR-76-372, The MITRE Corporation, Bedford, MA, 1977.
[CMADIII 95]	Report on Third Annual Workshop on Computer Misuse and Anomaly Detection, UC Davis, 10-12 January 1995. @ http://www.cs.purdue.edu/homes/swlodin/cmad_report.html
[DAC 96]	National Computer Security Center, *Discretionary Access Control Issues in High Assurance Secure Database Management Systems,* NCSC Technical Report-005, Volume 5/5, May 1996.
[DeWolfe 79]	DeWolfe, J. B., Szulewski, P., *Final Report of the 1979 Summer Study on Air Force Computer Security,* R-1326, C. S. Draper Laboratory, Inc., Cambridge, MA, October 1979.
[DoD 84]	Department of Defense, *Industrial Security Manual for Safeguarding Classified Information,* DoD 5220.22-M, Washington, DC, March 1984.
[DoD 85]	Department of Defense, *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28-STD, Washington, DC, December 1985.
[Entity 96]	National Computer Security Center, *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems,* NCSC Technical Report-005, Volume 2/5, May 1996.
[Filsinger 93]	Filsinger, J., ‘‘Integrity and the Audit of Trusted Database Management Systems,” *Database Security, VI: Status and Prospects: Results of the Sixth IFIP Working Group Conference on Database Security, Vancouver, Canada -21 August, 1992,* M. B. Thuraisingham and C. E. Landwehr editors, Elsevier Science Publishers, 1993.
[Gluck 93]	Gluck, F. B., ‘‘An Open Security Architecture,” *Proceedings of the 16th National Computer Security Conference,* October 1993.
[Graubart 84a]	Graubart, R. D., ‘‘The Integrity Lock Approach to Secure Database Management,” *Proceedings of the 1984 Symposium on Security and Privacy,* Oakland, CA, April 1984.

[Graubart 84b] Graubart, R. D., Duffy, K. J., ‘‘Design Overview for Retrofitting Integrity-Lock Architecture onto a Commercial DBMS,’’ *Proceedings of the 1985 Symposium on Security and Privacy,* Oakland, CA, April 1984.

[Halme 95] Halme, L. R., Bauer, R. K., ‘‘AIN’T Misbehaving -- A Taxonomy of Anti-Intrusion Techniques,’’ *Proceedings of the 19th National Information System Security Conference,* October 1995

[Hamilton 92] Hamilton, D., ‘‘Application Layer Security Requirements of a Medical Information System,’’ *Proceedings of the 15th National Computer Security Conference,* October 1992.

[Haigh 90] Haigh, J. T., O’Brien, R. C., Stachour, P. D., Toups, D. L., ‘‘The LDV Approach to Security,’’ *Database Security, III: Status and Prospects,* ed. D. L. Spooner and C. Landwehr, North-Holland, Amsterdam, 1990.

[Hinke 89] Hinke, T., et al., ‘‘A Layered TCB Implementation versus the Hinke-Schaefer Approach,’’ *Database Security III, IFIP Workshop on Database Security,* Monterey, CA, 1989.

[Hosmer 90] Hosmer, H. H., ‘‘Handling Security Violations Within an Integrity Lock DBMS,’’ *Database Security, III: Status and Prospects:,* D. L. Spooner and C. E. Landwehr editors, Elsevier Science Publishers, 1990.

[Hosmer 94] Hosmer, H. H., ‘‘Security Audit Research in Relational MLS DBMS,’’ Presented at the Government Workshop on the Status of Multilevel Secure RDBMSs, Southwest Harbor, Maine, June 1994.

[Hsieh 93] Hsieh, D., Lunt, T., Boucher, P., *The Sea View Prototype Final Report.* Technical Report A012, Computer Science Laboratory, SRI International, Menlo Park, CA, Aug. 1993.

[Inference 96] National Computer Security Center, *Inference and Aggregation Issues in Secure Database Management Systems,* NCSC Technical Report-005, Volume 1/5, May 1996.

[INFORMIX 94] *INFORMIX OnLine Secure Final Evaluation Report,* CSC-EPL-93/004, C-Evaluation Report No. 32/94, Library No. S-241,835, National Computer Security Center, 21 March 1994.

[Interp 95] The Interpreted TCSEC Requirements, National Computer Security Center, Dockmaster Criteria Forum, January 12, 1995.

[Jajodia 89] Jajodia, S., Gadia, S. K., Bhargava, G., Sibley, E., ‘‘Audit Trail Organization in Relational Databases.,’’ *Proceedings 3rd IFIP WG 11.3 Working Conference on Database Security,* September 1989.

[Jajodia 90] Jajodia, S., ‘‘Tough Issues: Integrity and Auditing in Multilevel Secure Databases,’’ *Proceedings of the 13th National Computer Security Conference,* Washington D. C., October 1990.

[Kang 94] Kang, M.H., Froscher, J.N., McDermott, J.P., Costich, O.L., and Peyton, R. ‘‘Achieving database security through data replication: the SINTRA prototype”. Proceedings of the 17th National Computer Security Conference, Baltimore, MD, September 1994.

[Knode 89] Knode, R.B., and Hunt, R.A., ‘‘Making Databases Secure with TRUDATA Technology,” *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December, 1989.

[Kogan 91] Kogan, B., Jajodia, S., ‘‘An Audit Model for Object-Oriented Databases,” *Proceedings of the 7th Annual Computer Security Applications Conference*, 1991.

[Lichtinan 90] Lichtman, A., Kimmins, J., ‘‘An Audit Trail Reduction Paradigm Based on Trusted Processes,” *Proceedings of the 13th National Computer Security Conference*, October 1990.

[O’Connor 89] O’Connor, J. P., et al., *Secure Distributed Database Management System Architecture.* RADC-TR-89-314, Volume I, December 1989.

[Oracle 94a] Oracle Corporation, *Trusted Oracle7 Technical Overview,* White Paper, Part A14774, January 1994.

[Oracle 94b] *Oracle7 and Trusted Oracle7 Final Evaluation Report,* CSC-EPL-94/004, C-Evaluation Report No. 07-95, Library No. S242,198, National Computer Security Center, 5 April 1994.

[Picciotto 87] Picciotto, J., ‘‘The Design of an Effective Auditing Subsystem,” *Proceedings of the IEEE Symposium on Security and Privacy,* April 1987.

[Poly 96] National Computer Security Center, *Polyinstantiation Issues in Multilevel Secure Database Management Systems,* NCSC Technical Report-005, Volume 3/5, May 1996.

[Schaefer 89a] Schaefer, M., Hubbard, B., Sterne, D., Haley, T. K., McAuliffe, J. N., Wolcott, D., ‘‘Auditing: A Relevant Contribution to Trusted Database Management Systems,” *Proceedings of the Fifth Annual Computer Security Applications Conference,* Tucson, Az, 1989.

[Schaefer 89b] Schaefer, M., Hubbard, B., Sterne, D., Haley, T. K., McAuliffe, J. N., Wolcott, D., *Secure DBMS Auditor: Final Technical Report and Functional Specification,* TIS Report #278, prepared for RADC by TIS under Contract No. F30602-87-D-0093, 28 December 1989.

[Schaefer 94] Schaefer, M., Smith, G., Halme, L., Landoll, D., *Assured DAC for Trusted RDBMSs Final Report,* ATR-94020, Arca Systems, Columbia, MD, September 1994. (Portions to be reprinted in IFIP 1995)

[Schaen 91] Schaen, S. I., McKenney, B. W., ‘‘Network Auditing: Issues and Recommendations,” *Proceedings of the 7th Annual Computer Security Applications Conference,* 1991.

[Sibert 88] Sibert, W. O., ‘‘Auditing in a Distributed System: SunOS MLS Audit Trails,” *Proceedings of the 11th National Computer Security Conference,* October 1988.

[Stang 93] Stang, D., Moon, S., *Network Security Secrets,* IDG Books, 1993.

[Schaefer 94] Schaefer, M., Smith, G., Halme, L., Landoll, D., *Assured DAC for Trusted RDBMSs Final Report*, ATR-94020, Arca Systems, Columbia, MD, September 1994. (Portions to be reprinted in IFIP 1995)

[Schaen 91] Schaen, S. I., McKenney, B. W., ‘‘Network Auditing: Issues and Recommendations,’’ *Proceedings of the 7th Annual Computer Security Applications Conference*, 1991.

[Sibert 88] Sibert, W. O., ‘‘Auditing in a Distributed System: SunOS MLS Audit Trails,’’ *Proceedings of the 11th National Computer Security Conference,* October 1988.

[Stang 93] Stang, D., Moon, S., *Network Security Secrets,* IDG Books, 1993. [TDI 91] Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-021, National Computer Security Center, April 1991.

[TDI 91] Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-021, National Computer Security Center, April 1991.

[TNI 87] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria,* NCSC-TG-D05, Version-I, National Computer Security Center, 31 July 1987.

[Williams 93] Williams, J. G., LaPadula, L. J., ‘‘Automated Support for External Consistency,’’ *The Computer Security Foundations Workshop VI* Franconia, IEEE, June 1993.

[Winkler-Parenty 90] Winkler-Parenty, H. B., ‘‘Sybase: The Trusted Subject DBMS,’’ *Proceedings of the 13th National Computer Security Conference*, Washington, DC, 1990.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1996	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Auditing Issues in Secure Database Management Systems	5. FUNDING NUMBERS
6. AUTHOR(S)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Security Agency Attn: V21, Partnerships and Processes Fort George G. Meade, MD 20755-6000	8. PERFORMING ORGANIZATION REPORT NUMBER NCSC Technical Report - 005 Volume 4/5 Library No. S-243,039
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report is the fourth of five companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses auditing issues in secure database management systems.

14. SUBJECT TERMS Auditing; Secure Database Management Systems	15. NUMBER OF PAGES 68
	16. PRICE CODE

17. SECURITY CLASSIFCATION OF REPORT Unclassified	18. SECURITY CLASSIIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT