Trusted RUBIX™ Version 6

Trusted Facility Manual
Revision 9

RELATIONAL DATABASE MANAGEMENT SYSTEM
Infosystems Technology, Inc.
4 Professional Dr - Suite 118
Gaithersburg, MD 20879
TEL +1-202-412-0152
Table of Contents

Chapter 1  INTRODUCTION .............................................................................................................. 1
  1.1 DOCUMENT ORGANIZATION ......................................................................................... 1
  1.2 DEFINITIONS .................................................................................................................. 2
  1.3 ACRONYMS .................................................................................................................... 5

Chapter 2  DATABASE ARCHITECTURE .................................................................................... 6
  2.1 PROCESS ARCHITECTURE ............................................................................................. 6
    CLIENT – SERVER MODEL .................................................................................................. 7
    DISPATCHER PROCESS ..................................................................................................... 7
    MAINTENANCE SERVER ................................................................................................. 7
  2.2 SUBSYSTEMS .................................................................................................................. 9
  2.3 TRUSTED RUBIX CLIENT/SERVER COMMUNICATION ...................................................... 9
  2.4 TRUSTED RUBIX SECURITY ARCHITECTURE ................................................................. 10

Chapter 3  SECURITY COORDINATION .................................................................................. 12
  3.1 TRUSTED RUBIX TOS DEPENDENCIES ....................................................................... 12
  3.2 USER IDENTIFICATION AND AUTHENTICATION .......................................................... 13
  3.3 SUBJECT BINDING ........................................................................................................ 13
    3.3.1 USER SECURITY ATTRIBUTE DEFINITIONS ....................................................... 14
    3.3.2 USER-SUBJECT BINDING ..................................................................................... 14
  3.4 DISCRETIONARY ACCESS CONTROL ............................................................................ 14
  3.5 MANDATORY ACCESS CONTROL .................................................................................. 14
    3.5.1 RELATIONSHIP TO THE TOS MAC POLICY ............................................................ 15
  3.6 TOS SECURITY ADMINISTRATOR DUTIES .................................................................... 15
  3.7 ROLE BASED ACCESS CONTROL .................................................................................. 16
  3.8 SECURITY AUDIT ........................................................................................................... 17
  3.9 OBJECT REUSE ............................................................................................................. 17
  3.10 RESIDUAL INFORMATION PROTECTION ...................................................................... 18
  3.11 PROTECTION OF SECURITY FUNCTIONS .................................................................... 18
  3.12 DOMAIN SEPARATION AND PROTECTION MECHANISM ............................................ 19
  3.13 INTERNAL TRANSFER OF DATA .................................................................................. 19
  3.14 ABSTRACT MACHINE TESTING ............................................................................... 20
  3.15 TOS RECOVERY FROM FAILURE ............................................................................... 20
  3.16 SECURE USAGE ASSUMPTIONS .................................................................................... 21
  3.17 SECURE OPERATIONAL AREA NETWORK .................................................................... 21

Chapter 4  TRUSTED FACILITY MANAGEMENT ....................................................................... 22
  4.1 USER MANAGEMENT ...................................................................................................... 22
    4.1.1 USER SECURITY POLICY ....................................................................................... 22
    4.1.2 ADMINISTRATIVE ROLES ....................................................................................... 23
    4.1.3 LISTING INFORMATION ABOUT USERS ................................................................. 24
  4.2 DATA SECURITY POLICY ................................................................................................. 25
  4.3 DISCRETIONARY ACCESS CONTROL (DAC) ................................................................. 25
  4.4 MANDATORY ACCESS CONTROL (MAC) ....................................................................... 27
    4.4.1 ROW POLYINSTANTIATION .................................................................................... 28
  4.5 SECURITY AUDIT ........................................................................................................... 29
  4.6 TRUSTED RECOVERY ..................................................................................................... 30
    4.6.1 WRITE-AHEAD LOGGING ....................................................................................... 30
    4.6.2 TRANSACTION FAILURE AND RECOVERY ............................................................ 30
    4.6.3 SYSTEM FAILURE AND RECOVERY ..................................................................... 31
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>4.6.4 MEDIA FAILURE AND RESTORE</strong></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td><strong>4.6.5 BACKUP</strong></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td><strong>4.6.6 RESTORE</strong></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td><strong>4.7 ASSURANCE</strong></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td><strong>4.7.1 CONFIGURATION MANAGEMENT</strong></td>
<td>33</td>
</tr>
<tr>
<td></td>
<td><strong>4.7.2 FULLY DEFINED EXTERNAL INTERFACES</strong></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td><strong>4.7.3 SECURITY ENFORCING HIGH LEVEL DESIGN</strong></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td><strong>4.7.4 SECURITY FEATURES USER GUIDE (SFUG)</strong></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td><strong>4.7.5 LIFE CYCLE SUPPORT</strong></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td><strong>4.8 SECURITY-RELEVANT EVENTS</strong></td>
<td>35</td>
</tr>
<tr>
<td></td>
<td><strong>4.9 DOCUMENTATION</strong></td>
<td>35</td>
</tr>
<tr>
<td></td>
<td><strong>Chapter 5</strong> TRUSTED RUBIX AUTHORIZATIONS AND ROLES**</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td><strong>5.1 AUTHORIZATIONS</strong></td>
<td>37</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.1 TRUSTED RUBIX MAC AUTHORIZATIONS</strong></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.2 TRUSTED RUBIX DAC AUTHORIZATIONS</strong></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.3 TRUSTED RUBIX AUDIT AUTHORIZATIONS</strong></td>
<td>41</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.4 TRUSTED RUBIX RESTORE AUTHORIZATIONS</strong></td>
<td>42</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.5 TRUSTED RUBIX ADMINISTRATIVE AUTHORIZATIONS</strong></td>
<td>43</td>
</tr>
<tr>
<td></td>
<td><strong>5.1.6 TRUSTED RUBIX USER AUTHORIZATIONS</strong></td>
<td>43</td>
</tr>
<tr>
<td></td>
<td><strong>5.2 TRUSTED RUBIX DEFAULT ROLES</strong></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td><strong>5.2.1 AUDIT ADMINISTRATOR</strong></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td><strong>5.2.2 DATABASE ADMINISTRATOR</strong></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td><strong>5.2.3 OPERATOR</strong></td>
<td>45</td>
</tr>
<tr>
<td></td>
<td><strong>5.2.4 SECURITY ADMINISTRATOR</strong></td>
<td>45</td>
</tr>
<tr>
<td></td>
<td><strong>5.2.5 USER AUTHORIZATIONS</strong></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td><strong>5.3 OPERATIONS ASSOCIATED WITH DEFAULT TRUSTED RUBIX ROLES</strong></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td><strong>5.4 OVERVIEW OF SOLARIS ROLE BASED ACCESS CONTROL</strong></td>
<td>49</td>
</tr>
<tr>
<td></td>
<td><strong>5.4.1 MANIPULATION OF AUTHORIZATIONS, EXECUTION ATTRIBUTES, AND RIGHTS</strong></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td><strong>5.4.2 SOLARIS SECURITY DATABASE FILES</strong></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td><strong>5.4.3 USING THE SOLARIS MANAGEMENT CONSOLE GUI</strong></td>
<td>52</td>
</tr>
<tr>
<td></td>
<td><strong>Chapter 6</strong> DATABASE ADMINISTRATION**</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td><strong>6.1 BULK LOADING</strong></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td><strong>6.1.1 SAME LABEL LOAD</strong></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td><strong>6.1.2 DIFFERENT LABEL LOAD</strong></td>
<td>55</td>
</tr>
<tr>
<td></td>
<td><strong>6.2 RECOVERY</strong></td>
<td>55</td>
</tr>
<tr>
<td></td>
<td><strong>6.2.1 BACKGROUND</strong></td>
<td>55</td>
</tr>
<tr>
<td></td>
<td><strong>6.2.2 LOGGING</strong></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td><strong>6.2.3 TRANSACTION RECOVERY</strong></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td><strong>6.2.4 SYSTEM RECOVERY</strong></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td><strong>6.2.5 MEDIA RESTORE (RXRESTORE)</strong></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td><strong>6.3 BACKUP</strong></td>
<td>57</td>
</tr>
<tr>
<td></td>
<td><strong>6.3.1 LATEST CONSISTENT MOMENT</strong></td>
<td>58</td>
</tr>
<tr>
<td></td>
<td><strong>6.3.2 LOG BOOK</strong></td>
<td>58</td>
</tr>
<tr>
<td></td>
<td><strong>6.3.4 FULL DATABASE BACKUP</strong></td>
<td>59</td>
</tr>
<tr>
<td></td>
<td><strong>6.3.5 RESTORE LOGGING</strong></td>
<td>59</td>
</tr>
<tr>
<td></td>
<td><strong>6.4 RESTORE</strong></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td><strong>6.5 ROW RECLASSIFICATION</strong></td>
<td>62</td>
</tr>
<tr>
<td></td>
<td><strong>6.6 MISCELLANEOUS</strong></td>
<td>62</td>
</tr>
<tr>
<td></td>
<td><strong>6.6.1 DROP, LIST, OR RENAME A DATABASE</strong></td>
<td>62</td>
</tr>
<tr>
<td></td>
<td><strong>6.6.2 DATABASE STORAGE LOCATIONS</strong></td>
<td>63</td>
</tr>
</tbody>
</table>

© 2014 Infosystems Technology, Inc.
# Table of Contents

**Chapter 7**  
AUDIT ADMINISTRATION ................................................................. 64  
7.1 AUDITED EVENTS ........................................................................ 64  
7.2 TRUSTED RUBIX AUDIT EVENTS .................................................. 65  
7.3 AUDIT DATA ................................................................................ 68  
7.3.1 ACL INFORMATION ................................................................. 71  
7.4 AUDIT ADMINISTRATION (RXAUDITSET AND RXAUDITRPT) .......... 71  
7.4.1 PRE-SELECTION CRITERIA ........................................................ 72  
7.4.2 POST-SELECTION CRITERIA ...................................................... 74  
7.4.3 AUDIT LOG FAMILIARIZATION ................................................. 75  
7.4.4 MAINTENANCE OF AUDIT DATA .............................................. 78  

**Chapter 8**  
DELIVERY AND INSTALLATION ......................................................... 79  
8.1 DELIVERY STRATEGIES .............................................................. 79  
8.1.1 ATTACK STRATEGIES ............................................................. 79  
8.1.2 PROTECTION TECHNOLOGIES ............................................... 79  
8.1.3 DELIVERY .............................................................................. 80  
8.1.4 INTRA-OFFICE DELIVERY ....................................................... 80  
8.1.5 VERIFIED INSTALLATION ....................................................... 80  
8.1.6 MESSAGE DIGEST DELIVERY ............................................... 80  
8.1.7 CUSTOMER REQUEST ............................................................ 81  
8.1.8 MESSAGE DIGEST VERIFICATION ......................................... 81  
8.2 INSTALLATION .......................................................................... 81  

**Chapter 9**  
Trusted RUBIX Configuration ........................................................... 82  

INDEX ................................................................................................. 90
CHAPTER 1  INTRODUCTION

The Trusted Facility Manual (TFM) is primarily designed for the security and administrative personnel managing the trusted facility. It describes functions and privileges to be controlled when running a trusted facility, details the security policy, its implementation procedures, audit mechanisms and various rights, restricted functions, and roles associated with the secure operation of Trusted Rubix. The TFM provides guidelines on the consistent and effective use of the TR protection features and its interaction with the trusted operating system.

In order to effectively administer TR, it is advisable for all security administrative personnel to be thoroughly familiar with the following documents.

1. The trusted operating system's Trusted Facility Manual (TFM) or equivalent
2. The trusted operating system's Security Features User's Guide (SFUG) or equivalent
4. SQL-ODBC Tutorial
5. SQL Reference Guide
6. ODBC Guide
7. Information Schema Guide
8. Administrative Commands Reference Guide
9. Security Policy Manager Tutorial
10. Security Policy Manager Reference Guide
11. SELinux Reference Guide

To assist the persons responsible for TR administration in becoming familiar with the architecture of TR, the following chapters present a brief description of what comprises a TR database server, how users communicate with the database server, and the TR architecture.

1.1 Document Organization

The TFM is organized in a logical sequence and each chapter addresses a specific topic:

→ Chapter 2 – outlines the client server architecture for TR. This includes an overview of the major subsystems of the TR server and client/server communication.

→ Chapter 3 – outlines coordination of TR security policy and functions with those of the trusted operating system.

→ Chapter 4 – outlines mechanisms by which authorized users may specify how to enforce the security policy of their organization.

→ Chapter 5 – describes how TR utilizes the RBAC mechanism provided by the trusted operating system to give users the ability to execute special security operations or supersede the MAC and/or DAC security policies. It provides examples of how TR authorizations can be managed through the trusted operating system. A set of default TR roles is also described.
Chapter 6 – describes standard database administration functions that need to be performed as part of the secure administration of TR. These include bulk loading, recovery, backup, row reclassification, and miscellaneous tasks.

Chapter 7 – describes audit mechanisms of TR and lists all auditable events and their configurations. It also addresses audit analysis and reporting functions.

Chapter 8 – provides a step-by-step description of Trusted Rubix delivery and installation procedures.

Chapter 9 – provides a complete description of the configurable parameters of the TR server.

1.2 Definitions

The following definitions have been provided to facilitate the reader's understanding of this document.

Access Control – the process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network). Synonymous with controlled access and limited access.

Authorized – possessing the rights and/or privileges necessary (in accordance with the Target of Evaluation (TOE) security policy) to perform an operation. A security policy defines a set of rules that regulate how assets are managed, protected, and distributed within a TOE.

Classification – a designation attached to information that reflects the damage that could be caused by unauthorized disclosure of that information. A classification includes a sensitivity level (UNCLASSIFIED, CONFIDENTIAL, SECRET, or TOP SECRET) and a set of zero or more compartments (CRYTO, NUCLEAR, etc.). The set of classifications, together with their hierarchical relation defining the allowed information flows between levels, form a lattice. Most dissemination controls, such as NATO, NOFORN, and NOCONTRACTOR, can be handled as additional compartment names.

Clearance – the degree of trust associated with a person. This is established on the basis of background investigations and the tasks performed by the person. It is expressed in the same way as classifications (i.e., a sensitivity level and a (possibly null) compartment set).

Confidentiality –
(1) The concept of holding sensitive data in confidence, limited to an authorized set of individuals or organizations.
(2) The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

Container – a multilevel information structure. A container has a classification and may contain objects (each with its own classification) and/or other containers.

Data Confidentiality – the state that exists when data is held in confidence and is protected from unauthorized disclosure.

Data integrity – the state that exists when computerized data is the same as that in the source documents and has not been exposed to accidental or malicious alteration or destruction.

Database – a database is a logical container that may contain many different kinds of data arranged in tables, where the data contained in each table is in some way related to the data in the other tables in that database.

Discretionary Access Control (DAC) – a means of restricting access to objects based on
the identity of the subjects and/or groups to which they belong. The controls are
discretionary in the sense that a subject with certain access privilege is capable of passing
that privilege (perhaps indirectly) on to any other subject (unless restricted by mandatory
access control).

- **Dominate** – security level (or sensitivity label) A is said to dominate security level (or
  sensitivity label) B if the hierarchical classification of A is greater than or equal to that of
  B and the nonhierarchical categories of A include all those of B as a subset.

- **Equal** – security levels (or sensitivity labels) are equal if the hierarchical level of both
  labels are equal and the nonhierarchical category sets are equivalent.

- **Evaluation Assurance Level (EAL)** – a package consisting of assurance components
  from Common Criteria (CC) Part 3 that represents a point on the CC predefined assurance
  scale. EALs provide an increasing scale that balances the level of assurance obtained with
  the cost and feasibility of acquiring that degree of assurance.

- **Incomparable** – security levels (or sensitivity labels) are incomparable if they are not
  equal and neither label is greater than the other.

- **Least Privilege** – principle that requires that each subject be granted the most restrictive
  set of privileges needed for the performance of authorized tasks.

- **Mandatory Access Control (MAC)** – a means of restricting access to objects based on
  the sensitivity (as represented by a label) of the information contained in the objects and
  the formal authorization (i.e., clearance) of subjects to access information of such
  sensitivity.

- **Multilevel Secure RDBMS** – a multilevel secure (MLS) RDBMS can store and process
  information at multiple security levels and serve multiple users, some of whom may not
  be cleared for all the information in the machine.

- **Multilevel Security (MLS)** – concept of processing information with different
  classifications and categories that simultaneously permits access by users with different
  security clearances, but prevents users from obtaining access to information for which
  they lack authorization.

- **Platform** – the combination of software, hardware, and/or firmware layers underlying the
  RDBMS.

- **RDBMS data** – data that the RDBMS creates, maintains, and uses to operate the RDBMS
  (e.g., configuration parameters, user security attributes, transaction log, audit instructions
  and records).

- **Relation** – a relation is a two dimensional (row and column) table of related data. Each
  table can contain different kinds of data.

- **Relational Database Management System** – a relational database management system
  (RDBMS) stores and manipulates data in the form of relations.

- **Resource** – anything used or consumed while performing a function. The categories of
  resources are:
    - Upper bound on:
      - time;
      - information,
      - objects (information containers),
      - or processors (the ability to use information).

  Specific examples are:
    - Upper bound on:
      - CPU time;
      - terminal connect time;
      - amount of directly addressable memory;
      - disk space;
      - number of disk requests per minute, etc.

- **Security Level** – the combination of a hierarchical sensitivity level and a set of non-
hierarchical categories that represents the sensitivity of information.

- **Security Policy** – a security policy defines a set of rules that regulate how assets are
managed

→ **Security Relevant Event** – any event that attempts to change the security state of the system, (e.g., change discretionary access controls, change the security level of the subject, change user password, etc.). Also, any event that attempts to violate the security policy of the system, (e.g., too many attempts to login, attempts to violate the mandatory access control limits of a device, attempts to downgrade a file, etc.).

→ **Sensitivity Label** – a piece of information that represents the security level of an object that describes the sensitivity (e.g., classification) of the data in the object. Sensitivity labels are used by the TOE security functions as the basis for mandatory access control decisions.

→ **Sensitivity Level** – a set of classifications (UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP SECRET) together with their hierarchical relation defining the allowed information flows between levels.

→ **Subject** – an active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state. Technically, a process/domain pair.

→ **Target of Evaluation (TOE)** – an IT product or system and its associated administrator and user guidance documentation that is the subject of an evaluation. The entity that is evaluated as defined by the Security Target (ST). While there are cases where a TOE makes up the entire product, this need not be the case. The TOE may be a product, a part of a product, a set of products, a unique technology never to be made into a product, or combinations of all of these, in a specific configuration or set of configurations. This specific configuration or set of configurations is called the evaluated configuration. The ST clearly describes the relation between the TOE and any associated products.

→ **Trusted Computing Base (TCB)** – the totality of protection mechanisms within a computer system, including hardware, firmware, and software. The combination of these components is responsible for enforcing a security policy.

→ **Trusted Rubix Default Roles** – a set of default roles installed and configured by TR during installation.

  - ↑ **Audit Administrator** (AUD) – a RDBMS administrative role that is limited to those privileges necessary to administer and review the RDBMS audit trail.

  - ↑ **Database Administrator** (DBA) – a RDBMS administrative role that is explicitly granted all discretionary access privileges to manage RDBMS objects in the RDBMS covered by the RDBMS discretionary access control policy.

  - ↑ **Database Operator** (OP) – a database administrative role whose privileges are limited to those privileges necessary to operate the RDBMS.

  - ↑ **Security Administrator** (SA) – a RDBMS administrative role that is explicitly granted the ability to cause an information flow for information covered by the RDBMS mandatory access control policy in order to perform administrative tasks operating on labeled information at multiple levels of security within the RDBMS.

  - ↑ **User** – an authorized RDBMS user who does not possess any trusted RDBMS administrative roles or authorizations (e.g. audit administrator, database administrator, database operator, security administrator), but does possess authorizations for exporting and importing data.
### 1.3 Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD</td>
<td>Audit Administrator default role</td>
</tr>
<tr>
<td>CC</td>
<td>Common Criteria</td>
</tr>
<tr>
<td>DAC</td>
<td>Discretionary Access Control</td>
</tr>
<tr>
<td>DBA</td>
<td>Administrator default role</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>EAL</td>
<td>Evaluation Assurance Level</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>ISQL</td>
<td>Interactive Structured Query Language</td>
</tr>
<tr>
<td>I&amp;A</td>
<td>Identification and Authentication</td>
</tr>
<tr>
<td>MAC</td>
<td>Mandatory Access Control</td>
</tr>
<tr>
<td>MLS</td>
<td>Multi Level Security</td>
</tr>
<tr>
<td>OP</td>
<td>Operator default role</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management Systems</td>
</tr>
<tr>
<td>SA</td>
<td>Security Administrator default role</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>ST</td>
<td>Security Target</td>
</tr>
<tr>
<td>TOE</td>
<td>Target of Evaluation</td>
</tr>
<tr>
<td>TOS</td>
<td>Trusted Operating System</td>
</tr>
<tr>
<td>TS/SCI</td>
<td>Top Secret/Sensitive Compartmental Information</td>
</tr>
</tbody>
</table>
CHAPTER 2 DATABASE ARCHITECTURE

Figure 1 Trusted RUBIX Client Server Architecture

Figure 1 shows the client server architecture for Trusted RUBIX. Each client is initiated by a TR user. The client process runs with the user’s credentials. When the client process connects to a specific database it requests a server from the TR Dispatcher. The Dispatcher caches active server processes. If a suitable cached server exists the Dispatcher allocates it to the client’s connection; otherwise, a server process is started with an exec system call by the TR Dispatcher. The server process sets its user credentials and security session sensitivity label and then performs operations on behalf of the client.

The two different user interfaces, RXISQL and ODBC, give the user flexibility in invoking the TR server. The RXISQL module provides an interactive SQL interface to submit ad hoc queries to the TR SQL Engine. The Open Database Connectivity (ODBC) module is an application programming interface (API) for database access. It is an implementation of Microsoft’s Open Database Connectivity (ODBC) specification.

2.1 Process Architecture

TR has adopted a client-server approach in which each user has both an application process and a RDBMS process – a process pair per user architecture. An invocation of TR consists of multiple cooperating processes. These processes consist of an application process, the Dispatcher process, a set of Maintenance Server processes, and a set of server processes.

WARNING

The TR process architecture assumes that it is running upon a targeted trusted operating system which provides a high degree of separation between security domains.
Client – Server model

When a user initiates a login session on the trusted operating system, processes operating on behalf of that user are assigned subject attributes that reflect the identity of the user (User ID and Group ID) and the trustworthiness of the user (process session security sensitivity label). When the user wishes to perform operations with Trusted Rubix he must first initiate a client process, which inherits the authentication attribute of the user. When the client process communicates with a TR privileged server process, the TR server process (rxserver process) uses the authentication attributes supplied by the trusted operating system to identify the user. The TR server process then stores the user’s authentication attributes and changes its own subject attributes to a special set of attributes that allow the TR server to access the database objects, perform Mandatory Access Control (MAC) and Discretionary Access Control (DAC) checks for existing objects, assign sensitivity labels to new database objects, and identify a user’s actions in the audit trail.

The application process consists of user application code linked with TR client software. The TR client software allows the application to communicate with the TR Server.

Dispatcher process

The TR Dispatcher (rxdsppr process) is used to service connection requests to server processes (rxserver processes) on behalf of rxisql and ODBC clients. It does this by either mapping the connection request to an idle, instantiated server process or instantiating a new server process. Clients are mapped to idle server processes that have the same security attributes as the client. The re-use of idle server processes dramatically reduces the time required to perform a database connection. Idle servers that surpass a configurable (via the rxconfig file) idle timeout are terminated. The Dispatcher executes as a standard UNIX service and is started and terminated by an administrator using the rxsvrman command or the UNIX service command. The UNIX service command uses the rxsvrman command to start the dispatcher process. The rxsvrman command is the seminal (and only) source for the necessary system high sensitivity label and the rubix/rubits effective user/group ID of the TR Dispatcher and the TR server. This insures that the Dispatcher and server processes cannot be instantiated by some other means by a malicious user. Note that the server does not trust the Dispatcher in any way. For more information on the rxsvrman command please see the Administrative Commands Reference Guide. For more information on the configuration of the TR Dispatcher and server please see Chapter 9 Trusted RUBIX Configuration in this document.

Maintenance Server

The TR Maintenance Server (rxmsvr process) is used to perform periodic background maintenance for a specific database. There may only be a single Maintenance Server instantiated at any given time for any specific database. The Maintenance Server process for a specific database may be explicitly instantiated or terminated using the rxdb command. Additionally, if configured in the rxconfig file, the Maintenance Server may be automatically started when a database server is started. Services provided by the Maintenance Server are:

- Background flushing of updated data pages
- Physical deallocation of dropped tables

Each service is configurable (e.g., service on/off, frequency of service) via the rxconfig file. For
more information on the `rxdb` command please see the Administrative Commands Reference Guide. For more information on the configuration of the TR Maintenance Server please see Chapter 9 Trusted RUBIX Configuration in this document.
2.2 Subsystems

Figure 2 shows the major subsystems of the TR server. The Server Interface Subsystem is a remote procedure call interface used by the client subsystems to submit operations on databases. It provides all TRUSTED RUBIX server external interfaces and allows clients to connect and disconnect to specific databases, start and terminate transactions, manipulate savepoints, and execute SQL operations. The Server Interface Subsystem executes SQL operations by parsing the text string representing the operation into an executable query tree. It submits these operations to the SQL Engine Subsystem, which accepts operations in the form of an executable query tree. The query trees are optimized for performance and mapped into operations on individual record files. The SQL Engine Subsystem checks for needed DAC privileges before submitting the operations on record files to the Kernel Subsystem. The Kernel Subsystem accepts record oriented operations from the SQL Engine Subsystem on Btree and data files. The Kernel Subsystem is responsible for enforcing all MAC restrictions on data objects. Thus, no MAC decisions are made above this subsystem. This subsystem also provides low level transaction and database operations. The Kernel Subsystem interacts with the Common Server Subsystem which in general interacts with the trusted operating system. The Common Server Subsystem provides functions that require shared memory to operate efficiently, such as the main memory page buffering mechanism.

![Diagram of Trusted RUBIX High Assurance Architecture]

2.3 Trusted RUBIX Client/Server Communication

Upon initiation of a connection request by a client to the dispatcher, a server process is invoked on a host machine. The host machine may be local or remote to the client machine. If the user specifies the connection should be made to a database residing on a specific host, via the DBNAME[@HOST_NAME:PORT] notation, a server process is invoked on the named host. If HOST_NAME is omitted, the local host is assumed.
The client, dispatcher, and server processes communicate with port based socket connections. A single statically pre-allocated port must be reserved for the client, dispatcher, and server processes. The specification of the port is made in the `etc/rxconfig` file located in the TR install directory. Please consult the TR TOS specific Installation Instructions for more information.

Once a socket connection is initiated between the client and server processes, the server authenticates the client by obtaining the user-id, group-id, and security label as operating system dependent trusted attributes of the socket connection. The server initializes itself with the user's credentials and performs database operations on behalf of the client.

### 2.4 Trusted RUBIX Security Architecture

The security policy architecture of the Trusted RUBIX RDBMS is shown in Figure 3 and indicates the following:

- the location of policy access checks relative to the SQL Engine and RDBMS Kernel is represented by the position of the box labeled with the security policy name. Policy access checks are executed over operations and objects in the RDBMS module directly below it. The DAC policy operates over SQL Engine operations and objects while all other policies operate over RDBMS Kernel operations and objects. A policy being above another policy implies that it is evaluated after the other policy;
- the values each policy uses to calculate its security decision are enumerated in the rectangular box to the right of the box labeled with the security policy name;
- the basic characteristics of each security policy’s rules are listed under the security policy name;
- note that only security policies that operate directly on RDBMS objects are shown (the RBAC policy is not shown).
Figure 3: Trusted RUBIX Security Architecture
CHAPTER 3  SECURITY COORDINATION

To meet high assurance database requirements, Trusted Rubix has been ported to run atop trusted operating system (TOS) platforms (e.g., Trusted Solaris, Red Hat Enterprise Linux 6, CentOS 6, and Scientific Linux 6). The unique combined security functionality provided by the TOS and TR yields a true high assurance application environment. The security strengths provided in most of the TOS’s include:

- Support for “least privilege” access
- Consistent secure password usage
- Protection from malicious code
- Flexible system administrator privileges
- Full support for trusted path and secure channel mechanisms.

3.1 Trusted RUBIX TOS Dependencies

These security strengths of the underlying TOS are relied upon by TR to provide:

- User Identification and Authentication. (See Section 3.2).
- User-Subject Binding, when a user initiates a trusted or untrusted command or TR application (See Section 3.3).
- Discretionary Access Control (DAC), enforce DAC security policies (See Section 3.4).
- Mandatory Access Control (MAC), enforce MAC security policies (See Section 3.5).
- Security Administrator Duties (See Section 3.6).
- Role Based Access Control (RBAC), management through authorizations and roles (See Section 3.7).
- Security Audit, time stamping the occurrences of the audit events and store logs to audit the functions that TR does not duplicate (See Section 3.8).
- Object Reuse, ensure that any residual information content of any resource is unavailable to all TOS objects upon resource allocation/deallocation (See Section 3.9).
- Residual Information Protection, (see Section 3.10).
- Protection of Security Functions, (See Section 3.11).
- Domain Separation and Protection Mechanism, the mandatory security mechanisms of the TOS confine an application to unique security domains (See Section 3.12) that is strongly separated from other domains in the system. Applications may still misbehave, but the resulting damage can be restricted to that single security domain. This ability to confine security breaches is critical to controlling data flows in support of a system security policy.
- Internal Transfer of Data, database import (rximport) and export (rxexport) of data, (See Section 3.13).
- TOS Recovery from Failure (See Section 3.15).
3.2 User Identification and Authentication

**TR** has fully integrated User Identification and Authentication (I&A) and the management of the authorizations required to access administrative functionality with the services provided by the trusted operating system. **TR** uses the User ID supplied by the TOS to identify the user with which it is communicating. **TR** uses the current group (Group ID) to which the user belongs to determine group privileges.

A user logs onto the TOS, he/she is assigned a clearance sensitivity label. The clearance sensitivity label represents the maximum session sensitivity label within which the user may initiate sessions. The clearance sensitivity label is assigned to the user when the user’s account is created and corresponds to the maximum level of trustworthiness of that user. Sessions, generally representing a login shell (i.e., C-shell or K-shell) and all processes initiated from that shell, are assigned a session sensitivity label. While a user may have only one clearance sensitivity label at a given time, they may have multiple concurrent sessions operating at different session sensitivity labels. All programs and processes executed within the context of the session are assigned the session sensitivity label.

When a user logs on successfully to the TOS at a given session sensitivity label, the user is implicitly logged on to **TR** at the same sensitivity label. They may only become different if a privileged user changes the current TOS or **TR** session sensitivity label during an execution of **TR**. The **TR** session sensitivity label is assigned to all objects inserted into any database. It is also used to enforce all MAC policies. In effect, **TR** clears users within the database for the range of sensitivity labels for which the user has been cleared by the TOS. Unless otherwise noted, the term session sensitivity label implies the **TR** session sensitivity label. The user, however, still needs either certain database privileges or authorizations to perform any database operation. The use of TOS I&A avoids duplication of storage and administration of passwords, and eliminates a potent avenue of attacking the RDBMS. This also reduces the size of the Trusted Computing Base (TCB) without compromising security which is a critical issue in higher assurance evaluations.

3.3 Subject Binding

**TR** identifies authorized users of a particular **TR** database by relying on the previously performed Identification and Authentication (I&A) of the underlying trusted operating system. In other words, the process, initiated by the user, will have the User ID, Group ID, and current session sensitivity label as its subject security attributes.

**TR** is an SQL-based client/server multilevel secure (MLS) Relational RDBMS operating in a client-server architecture. There is one instantiation of the server for each active client. A given client and server pair can run on the same machine or on different machines connected via a network. All communication between client and server takes place on a single-level socket connection. The client process is an application program, which has been linked with the **TR**
client software to communicate with its associated TR server process, which resides on the same machine as the database to be protected. Currently, two types of clients are supported in TR:

1. Instantiations of an Interactive SQL (RXISQL) interface that provides ad hoc access to databases; and
2. User-developed applications utilizing the ODBC API.

3.3.1 User Security Attribute Definitions

When a user successfully establishes a login session on the trusted operating system, processes, operating on behalf of that user, are assigned user security attributes that reflect the identity of the user (i.e., User ID and Group ID) and the subject sensitivity label as the sensitivity label of the user (login) session. When a user wishes to perform operations with Trusted Rubix, the user must first initiate an Interactive SQL (RXISQL) or Open Database Connectivity (ODBC) client process. The client process is associated with the security attributes of the User ID, Group ID, and session sensitivity label.

3.3.2 User-Subject Binding

The client and server processes may reside on the same machine (local mode) or different machines (non-local mode). In each case the user-subject binding method is the same. When the client process initiates a connection to a specific database, a TR server process (rxserver) is provided by the TR Dispatcher (rxdspr). The server is instantiated by the TR Dispatcher which executes with the process’s User ID and Group ID attributes to Rubix and Rubixtp, respectively, and the process security level to System High (SYSTEM HIGH). These attributes were inherited from the rxsvrmn command which instantiated the dispatcher. The TR server process extracts the user’s security attributes from the socket connection and uses the stored user security attributes to identify the user and to perform Mandatory Access Control and Discretionary Access Control mediation on the RDBMS objects, and describe users in the audit trail. The stored user security attributes include:

- User ID
- Group ID
- Session sensitivity label

3.4 Discretionary Access Control

When a database application (client) communicates with TR, the User ID and current Group ID are supplied by the underlying TOS to identify the user it is communicating with. The User ID and Group ID are used to access the user’s DAC privileges, and therefore are the basis for the DAC decisions made by TR (see Section 4.3).

3.5 Mandatory Access Control

The database session label is used by TR to make MAC decisions. The database session label is initially set to the TOS session label when the user connects to the TR server. It may later be changed (only by the Security Administrator) using the ALTER SESSION command.

TR provides mandatory access control features that are built atop the mandatory access control primitives of the TOS. The security lattice defined for the TOS is used for the TR protected
subjects and objects. The TOS is relied on to protect the operating system’s named objects, which include directories, files, memory, devices, and sockets created and/or used by TR (i.e. audit record files, import/export and recovery files, database configuration files, and database files). For more information see Section 4.4.

### 3.5.1 Relationship to the TOS MAC policy

The platform selected for the implementation of Trusted Rubix assumes a Trusted Operating System (TOS). The kernel of the TOS is assumed to be isolated from the trusted processes operating in the user domain (and is verified as such at the time of implementation of TR on the TOS). Consequently, the TR Trusted Security Functions are integrated with the underlying TOS Functions.

In TR, all MAC protected data are stored in single-level operating system objects. To support fine-grained multilevel objects (e.g., rows), sensitivity labels are attached to individual database items within each operating system object. Note that these object sensitivity labels are TR sensitivity labels and not operating system sensitivity labels; the operating system views these object sensitivity labels strictly as data and attaches no security significance to them. TR is trusted to properly associate and maintain the object sensitivity label of each item and to correctly interpret those object sensitivity labels so that, in cooperation with the TOS kernel, the security policy can be correctly enforced.

The operating system performs MAC on its users and objects. TR provides MAC on RDBMS users and objects. The object sensitivity labels, i.e., the complete MAC security lattice, for TR MAC are provided by the trusted operating system.

**NOTE**

In addition to the Multilevel Security (MLS) Mandatory Access Control (MAC) policy, TR supports the Type Enforcement (TE) MAC policy of SELinux and a proprietary Attribute Based Accessed Control (ABAC) MAC policy of the Security Policy Manager (SPM). In general, all configured MAC policies must permit an operation for it to succeed. For more information on TE and SELinux please see the Trusted RUBIX SELinux Guide and for more information on ABAC and the SPM please see the Trusted RUBIX Security Policy Manager Reference Guide and Tutorial.

**WARNING**

The user who has been granted the authorizations equivalent to RDBMS Security Administrator must take extreme care in importing/exporting labeled data to and from the TOS. A significant coordination is required for mapping various labels. The imported rows should include the character label identifiers to ensure the data is labeled correctly and to allow the label to be verified visually by the RDBMS Security Administrator.

### 3.6 TOS Security Administrator Duties

TR relies on the TOS security administrators to:

- Grant TR users appropriate authorizations for different TR security management and
administrative roles.

→ Create the TR users’ accounts and assign users to the appropriate TR user groups.

→ Add authorizations / roles to the TOS security database, create directories to hold executable and data files, set privileges and sensitivity label on the directories, copy and assign privileges to executable programs (As performed during system installation, see Chapter 8).

→ Inform the TR Administrators when a user account is deleted.

TR integrates with the RBAC mechanism provided by the TOS to give users the ability to execute special trusted operations or supersed the MAC and/or DAC security policies. Once TRusted Rubix authorizations are added to the security database of the TOS they may be given or taken away in the same manner as other the TOS authorizations / roles.

The TR authorizations only have significance for operations on TR software (i.e., they imply no privilege for any other software). Authorizations are discussed in detail in Section 5.1.

For an authorization to be effective it must be present on the machine that contains the database being accessed.

3.7 Role Based Access Control

Role Based Access Control relies upon predefined authorizations to create mutually exclusive roles for each administrative task. By separating duties, security is increased for the following reasons:

→ Assign only needed role/right to users (Least privilege)
→ Simplified authorization management
→ Independent mappings:
  ↑ role-permission, ↑ user-role, and ↑ role-role relationships

The TRusted Computing Base (TCB) is tightly integrated with the TOS TCB. The TOS System Administrator should be familiar with granting authorizations and the creation of user roles.

The TOS site administrators may create roles with any combination of authorizations regardless of whether those authorizations are for TR or the TOS. TR supports full flexibility in the assignment of authorizations to roles. A role may contain many authorizations, or only a single one.

TR provides five default roles; each with a group of TR authorizations. These five default roles are given as follows:

→ RDBMS Audit Administrator (AUD)
→ Database Administrator (DBA)
→ Database Operator (OP)
→ RDBMS Security Administrator (SA)
→ Non-administrative Database User

The default security roles are discussed in Section 5.2 Trusted RUBIX Default Roles of this manual and the Trusted RUBIX SELinux Guide.

### 3.8 Security Audit

This section introduces TR auditing. Specific information about the binding of TR to the TOS can be found in Section 3.3. Detailed audit information can be found in Chapter 7.

TR is tightly integrated with the TOS, and relies upon it for both user and role management. Consequently, the auditor will need to refer to the TOS security logs to audit the functions that TR does not duplicate, such as the creation, modification, and destruction of user accounts, the granting and revoking of authorizations, the assignment of roles to particular users, and the assumption of particular roles by privileged users. TR maintains separate audit logs for all other user and administrative auditing. The audit log is a UNIX file secured by the TOS at SYSTEM HIGH. The TOS's strong security mechanisms prevent attackers from accessing and modifying the audit log.

#### WARNING

Consult the TOS administrative documentation and the Trusted RUBIX OS specific installation guide for details on administering the RBAC mechanism.

TR may selectively audit certain actions performed by particular users or groups. The User ID and Group ID supplied by the TOS identify the user communicating with TR. The same User ID and Group ID are used in the audit trail to identify the user in the audit trail as the user who initiated the auditable event.

The TOS is used to assign to a user or role the appropriate authorizations to permit them to use the *rxauditset* command. The *rxauditset* command is used to specify or display the criteria which determine whether an auditable event is eligible to be recorded by the TR audit trail.

### 3.9 Object Reuse

TR has only two basic objects on the server that may be reused, the segment (page) and the slot (record). All other server side object allocation relies on one or both of these basic object allocations. For example, a file is made up of its associated data pages and each table is stored as a file. The table’s structure and ACL information are stored in other Definition Schema tables. Catalogs and schemas are simply files that hold table, view, index, and schema (in the case of
catalogs) names. Thus, every storage object, other than a row, is stored as a file constructed of pages. Rows within a table correspond to a record in a page.

When a segment is allocated on the server its entire space is overwritten with zeros. Thus, each newly allocated segment or page starts out clean. New slots are allocated within a page when a slot is inserted into that page. Only the exact number of bytes necessary for the slot is allocated and the new slot data is immediately written into that specific allocated slot. It should be noted that due to the Multi-version Timestamp Ordering (MVTO) scheduling protocol Trusted Rubix objects are not physically deallocated until a garbage collection mechanism initiates the process. This initiation may be delayed for a potentially unbounded amount of time. Because the object reuse policy is only concerned with objects as they are reused, this delay in deallocation has no impact on the security aspects of the reuse policy.

This Residual Information Protection security function performs residual information protection on the server and ensures that any residual information content of any resource is unavailable to all objects upon the resource’s allocation. When TR drops a database all of the operating system files are removed. The TOS is relied upon to perform object reuse of the dropped files correctly.

All allocation and deallocation of data structures on the client side occur locally to the untrusted client process. Furthermore, these objects are not MAC controlled objects. Therefore, an object reuse policy is not needed for these objects and all of these functions are non-TSF.

### 3.10 Residual Information Protection

This Residual Information Protection security function performs residual information protection and ensures that any residual information content of any resources is unavailable to all objects upon the resource’s allocation. When TR drops a database all of the TOS files are removed. The TOS is relied upon to perform object reuse of the files correctly. TR has only two basic objects that may be reused, the page and the record within the page. All other object allocation relies on one or both of these basic object allocations. For example, a table is made up of its associated data pages. The table’s structure and ACL information are stored in other information schema tables. Catalogs and schemas are simply tables that hold table and view names. Thus, every storage object, other than a row, is a table constructed of pages. Rows within a table correspond to a record in a page. When a page is allocated its entire space is overwritten with zeros. Thus, each newly allocated segment or page starts out clean. New records within the page are allocated when a record is inserted into that page. Only the exact number of bytes necessary for the record is allocated and the new record data is immediately written into all of the bytes of that specific allocated record. It should be noted that due to the Multiversion Timestamp Ordering (MVTO) scheduling protocol, TR objects are not physically deallocated until a garbage collection mechanism initiates the process. This initiation may be delayed for a potentially unbounded amount of time. Because the object reuse policy is only concerned with objects as they are reused, this delay in deallocation has no impact on the security aspects of the reuse policy.

### 3.11 Protection of Security Functions

The reference monitor security function ensures that the TSF is always invoked (i.e., non-bypassability) before any functions are allowed to proceed. The TR client/server architecture is based on the TR server receiving requests from the client for operation on the database. The TR Server resides on the machine where the database is stored. Within the TR Server, the TR Server Interface subsystem is the client’s only entry point into the TOE security functions of TR.
(see Figure 2 in Section 2.2 for a diagram of the subsystems). The Server Interface subsystem handles \texttt{TR} client database operation requests and invokes the TSP enforcement security functions of the \texttt{TR} Server. Therefore, no database operation can proceed prior to the TSP enforcement functions having been invoked. The Server Interface subsystem invokes the SQL engine subsystem, which is responsible for enforcing DAC policies for operation on the database. The SQL Engine subsystem checks for needed DAC privileges before submitting the operations on record files to the Kernel subsystem. The Kernel subsystem is responsible for enforcing all MAC restrictions on data objects. The Kernel subsystem checks the needed MAC sensitivity label of the user before performing the low level transaction and database operations.

### 3.12 Domain Separation and Protection Mechanism

In most cases, the \texttt{TrustedRubix} client and the \texttt{TR} server reside on different machines. The \texttt{TR} server resides on the machine where the databases are kept and protected. \texttt{TR} stores its objects in TOS files using what is known as the container storage model. The most primitive storage structure for \texttt{TR} objects are TOS files. Each database is stored in multiple TOS files. All TOS files for a unique database are stored in a single TOS directory. The directories for all databases are stored in a single databases directory. Because \texttt{TR} uses this container storage model, the isolation and protection of \texttt{TR} file and directory objects are accomplished by using the TOS's DAC, MAC, and domain separation and protection mechanisms. A special \texttt{TR} User ID (RUBIX) and Group ID (RUBIXTP) is created and reserved for this purpose. Each database file and directory is owned by the user/group pair of RUBIX/RUBIXTP. The \texttt{TR} server, executing with user/group as RUBIX/RUBIXTP, views the database space as its own address space. For each of the directory and file objects, its permissions are set so only members of RUBIXTP group may access these objects. The \texttt{TR} executables are restricted to the RUBIXTP group. No login user is allowed in the RUBIXTP group. Thus, they cannot access any database files or directories that are owned by the user/group pair of RUBIX/RUBIXTP. These TOS directories and files of the databases are labeled at system high sensitivity label to offer additional protections. Thus, only \texttt{TR} executables labeled at system high may access the TOS directories and files that contain \texttt{TR} objects.

Under a single directory that is named "databases", there is one subdirectory (named the same as the database name) per database representing each existing database in \texttt{TR}. \texttt{TR} objects within a single database are stored in multiple TOS files in this subdirectory. \texttt{TR} treats each TOS file as a container. The \texttt{TR} objects such as databases, catalogs, schemas, tables, views, indexes, and records are created from the container. Each of these abstract objects is assigned a security label and an access control list (ACL). Each TOS file may hold information labeled at any sensitivity label. \texttt{TR} ensures the isolation and protection of \texttt{TR} objects in the TOS files and arbitrates subject access to each \texttt{TR} data object, which is associated with an ACL and a sensitivity label, based upon the MAC and DAC policies.

### 3.13 Internal Transfer of Data

\texttt{TR} uses the same MAC security lattice as the underlying TSO. However, there are occasions when the movement of protected data from one domain to the other needs to be understood.

Data moves from the TOS protected domain to the \texttt{TR} protected domain during an \texttt{INSERT}, \texttt{UPDATE}, and \texttt{rximport} operation. When the user is not operating as the \texttt{TR} Security Administrator, the \texttt{TR} session sensitivity label always equals the TOS session sensitivity label and \texttt{TR} automatically labels the data with its session sensitivity label. In this mode, there can be
no change in the labeling of the data as it changes from the TOS domain to the TR domain. When the user is operating as the Trusted Rubix Security Administrator, there is a potential for the data to change its sensitivity label and the user should act accordingly. Using the ALTER SESSION SET LABEL command via the RXSQL client, the TR Security Administrator may change the TR session label to one that is greater than, less than, or incomparable to the current TOS session sensitivity label.

If the TR Security Administrator is using the rximport command in the multilevel mode, the data will be explicitly labeled to that specified in the text import file.

Data moves from the TR protected domain to the TOS protected domain during a SELECT and rxexport operation. When the user is not operating as the TR Security Administrator, the TR session sensitivity label always equals the TOS session sensitivity label and TR returns only rows that are dominated by the TR session sensitivity label. Once the data enters the TOS domain, the TOS will implicitly label at the TOS session sensitivity label.

WARNING

This may result in an implicit upgrade of data from the current data’s TR object sensitivity label to the current TOS session sensitivity label.

Using the ALTER SESSION SET LABEL command via the RXSQL client, the TR Security Administrator may change the TR session label to one that is greater than, less than, or incomparable to the current TOS session sensitivity label.

WARNING

When the data crosses from the TR domain to the TOS domain the data will be implicitly re-graded from the current TR session sensitivity label to the current TOS session sensitivity label.

3.14 Abstract Machine Testing

The TOS automatically runs a suite of tests periodically during normal operation and at the request of an authorized administrator. This suite of tests demonstrates the correct operation of the security assumptions provided by the abstract machine that underlies the TSF.

3.15 TOS Recovery from Failure

After a failure or service discontinuity, TOS enters a maintenance mode where the ability to return the TOE to a secure state is provided.

The TOS ensures that the following Security Functions (SF) and failure scenarios have the property that the SF either completes successfully, or the indicated failure scenarios recover to a
consistent and secure state:

a) the SF checks whether a specified privilege is assigned to any role, but the database containing the privilege data is not on-line or the particular data table is inaccessible;

b) the SF checks whether a specified role has been assigned to a particular user, but the database containing the role membership information is not on-line or the particular data table is not inaccessible.

### 3.16 Secure Usage Assumptions

**WARNING**

It is assumed that database administrators, database operators, DBMS security administrators, and DBMS audit administrators are competent, and merit trust placed in them.

**WARNING**

It is assumed that authorized DBMS users are familiar with applicable DBMS security policies and procedures, and merit trust placed in them.

**WARNING**

It is assumed that the DBMS is protected against disasters such as loss of power, fire, flood, and destruction of facilities.

**WARNING**

It is assumed that the DBMS, host OS, and IT environment are protected from physical attack.

**WARNING**

It is assumed that the environment protects information while in transit between the DBMS and components of the IT environment.

### 3.17 Secure Operational Area Network

**Trusted Rubix** relies on the TOE environment for its client/server communications security, which includes cryptosecurity, transmission security, emission security, and physical security of communications material. Communications security material includes, but is not limited to, key, equipment, devices, documents, firmware or software that embodies or describes cryptographic logic and other items that perform communications security functions.
CHAPTER 4  TRUSTED FACILITY MANAGEMENT

**Trusted Rubix** provides mechanisms by which authorized users may specify how to enforce the security policy of their organization. The set of procedures, tools and attributes utilized to perform this function are the foundation for Trusted Facility Management. The administrative roles are trusted user defined packages of authorizations bundled in a manner such that they support the intent of the security policy.

4.1 User Management

Database users are the access path to the data stored in and controlled by **TR**. To support the high level of assurance required to manage user access, **TR** relies solely on the user management mechanism of by the TOS. Users communicate with the **TR** server via client programs such as database applications. When a database application communicates with **TR**, the security attributes (User ID, Group ID, authorizations, and session sensitivity label) supplied by the TOS are used by **TR** to identify the user. The security attributes are retrieved from the security attributes of the user’s client process.

The session sensitivity label is used to perform Mandatory Access Control decisions on behalf of the user. The User ID and Group ID are used to make Discretionary Access Control decisions by **TR**. The User ID and Group ID are also used in the **TR** audit trail to identify the user initiating the audit event. The TOS authorizations/roles are used to identify which trusted users are allowed to execute special **TR** administrative commands and which users may supersede the DAC and/or MAC security policies. The authorizations recognized by **TR** are created during the install process and are granted to users in the TOS security database.

Conversely, if a user loses his TOS account he loses his access path to the **TR** server, and hence to the data stored therein.

**WARNING**

Although the user’s access path is blocked when his/her TOS account is dropped, the database administrator is also required to reassign the user’s DAC privileges stored in the **TR** server (See Section 4.1.3 for information about accessing DAC privileges).

**TR** does not directly support the concept of object ownership, but rather assigns DAC privileges to the object’s creator. This simplifies the administrative job of removing a user to deleting a user’s DAC privileges, since no objects are directly associated to any user. Nevertheless, care must be taken to reassign certain of the DAC privileges (particularly those associated with database modification, especially GRANT).

**WARNING**

Only trusted users should have the privilege to create user accounts and to manage users.

4.1.1 User Security Policy

For all types of database users, the security issues of password management and privilege management apply.
If a database has many users, it may be beneficial to use the TOS roles mechanism when performing privilege management. Alternatively, for a database with only a few users, it may be simpler to manage privileges by assigning authorizations directly.

**TR** has two types of user, administrative users and normal users. Administrative users are those who hold administrative authorizations, and are thereby authorized to perform administrative tasks, such as database backup and restoration, audit and audit file management, and user DAC privilege management.

Normal users may hold the *rubix.user.* authorizations which permit them to use the bulk import and export utilities (See *rxexport* and *rximport* in the **TR** Administrative Commands Guide.

A special DAC privilege that Administrators may find useful is the NULL privilege. The NULL privilege is different from other DAC privileges in that it is subtractive and overrides all other DAC privileges. A user can be blocked from accessing a given object simply by granting him the NULL privilege on that object. This eliminates the task of evaluating a given user’s set of DAC privileges (including those inherited through group memberships) and then revoking each offending privilege.

In general, **TR** provides fine grained DAC privileges for each database object. While certain users, such as application developers may need to create tables, indexes, and views to perform their job functions, others users may only need full access, or even only read access, to tables and views.

Another feature of **TR** that is useful in administering MAC security policy is row level polyinstantiation. Row level polyinstantiation permits rows with identical primary keys to exist in the same table if they have different sensitivity labels (classification). This is a valuable tool in enforcing user security policy because it eliminates a covert channel for users to probe tables with, and provides a “cover story” mechanism for highly classified information. See Section 4.4.1 for more information on row polyinstantiation.

**WARNING**

Physical security must also be taken into account when considering user security. Many administrative commands can only be run from the server itself. Hence, physically securing the server may thwart a malicious user who has compromised the password policy, but cannot gain access to the console where the administrative commands must be run. Similarly, a periodic check for logged-in, but unattended client consoles will help to maintain system security.

### 4.1.2 Administrative Roles

The assurance objectives of the Common Criteria place two basic requirements on Trusted Facility Management:

→ Administrative roles of the system are recognized as essential to ensure that the security
The policy of a system is enforced.

→ The administrative roles are defined such that the system is assured to be implemented and operating correctly.

**Trusted Rubix** meets the Common Criteria administrative roles requirements via the use of a mechanism called Role Based Access Control (RBAC). RBAC provides a flexible, fine-grained way to package authorizations to meet particular job responsibilities (see Section 5.4 for further details on RBAC). This allows the Security Administrator to partition authorizations to create administrative roles that best meet the security requirements of their application. **TR** provides a default set of administrative roles that fulfill the traditional partitioning of administrative job responsibilities. The default **TR** administrative roles are built on the host as part of product installation. The default roles and their associated authorizations are discussed later in Section 5.2 of this document.

**4.1.3 Listing Information about Users**

**TR** ensures that a user has the required privilege and/or authorization information about each user’s DAC privileges which can be retrieved from the information schema. One view of user privileges exists for each object type. The views are named for the object with “_PRIVILEGES” appended. For example, the name of the view that returns data about the privileges held on the database object is “DATABASE_PRIVILEGES”. These views are stored in the “SYSTEM_CATALOG” and “INFO_SCHM” schema of each database. Details about the views and the information that can be retrieved through them are in the following table:

<table>
<thead>
<tr>
<th>View name</th>
<th>View Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_PRIVILEGES</td>
<td>“select grantor, grantee, grantee group, privilege type, is grantable from system_catalog.definition_schema.database_privileges table;”</td>
</tr>
<tr>
<td>CATALOG_PRIVILEGES</td>
<td>“select grantor, grantee, grantee group, catalog name, privilege type, is grantable from system_catalog.definition_schema.catalog_privileges table;”</td>
</tr>
<tr>
<td>SCHEMA_PRIVILEGES</td>
<td>“select grantor, grantee, grantee group, catalog name, schema name, privilege type, is grantable from system_catalog.definition_schema.schema_privileges table;”</td>
</tr>
<tr>
<td>TABLE_PRIVILEGES</td>
<td>“select grantor, grantee, grantee group, catalog name, schema name, table name, privilege type, is grantable from system_catalog.definition_schema.rx_table_privileges table;”</td>
</tr>
<tr>
<td>COLUMN_PRIVILEGES</td>
<td>“select grantor, grantee, grantee group, catalog name, schema name, table name, column name, privilege type, is grantable, remarks from system_catalog.definition_schema.rx_column_privileges table;”</td>
</tr>
<tr>
<td>RX_COLUMN_PRIVILEGES</td>
<td>“select * from system_catalog.definition_schema.rx_column_privileges table;”</td>
</tr>
</tbody>
</table>
View name | View Definition
---|---
RX_TABLE_PRIVILEGES | “select * from system_catalog.definition_schema.rx_table_privileges table;”

### 4.2 Data Security Policy

Data Security includes the mechanisms that control the access to and use of the database at the object level. Your data security policy determines which users have access to a specific database, catalog, schema, table, view, row, and column. Users can also be granted the privilege to issue certain SQL commands on specific columns or tables, such as SELECT, but denied the privilege to issue other commands, for example INSERT. Your security policy is determined primarily by the level of security your application requires and the sensitivity of the data.

The Data Security Policy enforced by **TR** includes MAC and DAC mechanisms based on the user's identity or group affiliation, and the TOS mediated roles for access to administrative commands. In addition, **TR** has an extensive audit capability to assist the administrator in monitoring database activity. The **TR** Audit capability is presented in Chapter 7. Discretionary Access Control (DAC) is intended for an environment where cooperating users process data at the same level of sensitivity labeling. DAC based on individual users and/or groups enables users to share access to objects in a controlled fashion. DAC is discussed in Section 4.3.

Mandatory Access Control (MAC) is intended for applications that manage classified data and enforce the mandatory data access policies. Sensitivity labels, constructed from hierarchical classification levels and non-hierarchical categories, are assigned to each subject and object. The integrity of the label is protected (see Chapter 4 of the **TR** Security Features User's Guide for a discussion of label management). **TR** provides mandatory access control features that are built atop the mandatory access control primitives of the TOS. The security lattice defined for the TOS is used for the **TR** protected subjects and objects. MAC is discussed in Section 4.4.

**TR**’s MAC security mechanism supports three levels of polyinstantiation. Polyinstantiation allows two rows of data with identical primary keys to exist in the same table if they have different object sensitivity labeling (classification) (e.g. were inserted by two users of differing subject sensitivity labeling (clearance)). This is a key capability for prevent user’s from inferring the existence of data that their clearance does not dominate, and also for providing a “cover story” for sensitive data. See Section 4.4.1 for additional information.

As noted in Section 4.1, **Trusted Rubix** does not directly support the concept of object ownership, but relies upon the relationship of the object’s sensitivity labeling and user labeling and privilege, or authorization to determine if a given action is permitted or not. When a user creates an object he/she is granted control of the object through the automatic granting of DAC privileges. The MAC and DAC privilege automatically granted to the object’s creator is analogous to the “object owner” construct of other database products. If another user acquires the same privilege on the object, he/she then is equivalent to the object’s creator.

### 4.3 Discretionary Access Control (DAC)

Discretionary Access Control (DAC) is a means of restricting access to objects based upon the identity of users and/or groups to which they belong. The controls are discretionary in the sense
that a user with a certain access privilege is capable of passing that privilege to any other user.

Operations on TR databases, catalogs, schemas, tables, indexes, views, table columns and view columns are controlled by DAC policy.

The discretionary access controls of TR are characterized in terms of subjects, named objects, and the operations which subjects can perform upon named objects. Subjects hold certain access privileges with respect to the named objects maintained by TR. Privileges for a named object are propagated either by a user holding the WITH GRANT OPTION on a privilege (initially only the object’s creator) or by a database administrator. The exception to this is the NULL privilege which overrides all other privileges and denies all access by the subject to the object. Because the NULL privilege removes all access to an object it has no corresponding WITH GRANT OPTION. To give the NULL privilege to someone the GRANTNULL privilege is required. The GRANTNULL privilege does have an associated WITH GRANT OPTION associated with it.

Discretionary security is enforced in TR by allowing users to specify which users and groups are authorized to perform specific operations on particular objects. Different access privileges control different operations. To modify privileges on an object, the user’s session sensitivity label must equal the object’s sensitivity label.

Each TR table has an access control list (ACL) that specifies the distribution of DELETE, SELECT(I), INSERT(I), UPDATE(I), REFERENCES(I), CRVIEW(I), REFVIEW(I), NULL(I), and GRANTNULL(I) privileges. Each TR view has an ACL that specifies the distribution of DELETE, SELECT(I), INSERT(I), UPDATE(I), CRVIEW(I), NULL(I), and GRANTNULL(I) privileges. The PRIVILEGE(I) form of these privileges permits the subject PRIVILEGE access to column “I” of a table or view.

Each TR database, catalog, and schema has an ACL that specifies the distribution of READ, WRITE, EXEC, NULL, and GRANTNULL privileges.

When calculating a user's effective privileges to the object, the NULL privilege negates all other privileges. The NULL privilege overrides all other privileges and explicitly denies all access by the subject to the object. For instance, if a user has the SELECT and NULL privileges on the columns of a table, the NULL privilege takes precedence and the user will not be able to select from the table. If the NULL privilege were revoked, the user would then have the ability to select from the table. Because the NULL privilege removes all access to an object it has no corresponding WITH GRANT OPTION. To grant the NULL privilege to someone, the GRANTNULL privilege is required. Giving the GRANTNULL privilege on an object gives a user the ability to deny others access to that object. The GRANTNULL privilege does have a WITH GRANT OPTION associated with it.

Each privilege, with the exception of the NULL privilege, has an associated GRANT privilege which specifies if the user may give the privilege away. The distribution of object access privileges to various users and groups of users is specified in the access control list (ACL) attached to the object. Each record in the ACL obeys one of the following three formats:

- U : User ID : maxprivils : column-bitmap
- G : Group ID : maxprivils : column-bitmap
- PUBLIC : maxprivils : column-bitmap

The U, G, or PUBLIC field designates the ACL type. The User ID and Group ID fields identify the user or group who is granted the privileges. The maxprivils field stores the sum of all privileges the particular user or group has to that object along with information on whether the user may grant
the privilege to other users. The column-bitmap field is used for tables and views. The various
column privileges are stored in this field along with information on whether the user may grant
the privilege to other users. It should be noted that there can only be a single ACL for any specific
User ID or Group ID.

It should also be noted that table rows are not DAC controlled objects in Trusted
Rubix. However table columns are DAC controlled objects. Users access table rows by
referencing the specified table and table columns in a query.

**TR** permits the **TR** DBA to operate with special authorizations (**rubix.dac.***) which allows for
superceding the DAC policy for specific operations. There are specific authorizations that allow
for INSERT, UPDATE, DELETE, SELECT, CREATE, DROP, GRANT, and REVOKE on objects without
regard for the DAC privileges. For more information on **TR** authorizations see Section 5.1

The **TR** DAC policy is described in Chapter 2 of the Security Features User's Guide.

### 4.4 Mandatory Access Control (MAC)

Mandatory security is enforced in **TR** by associating sensitivity labels with users and objects
mediating all accesses based on those sensitivity labels. The set of sensitivity labels is
partially ordered by the dominates relationship. The set of sensitivity labels and the dominates
relation form a lattice.

Users are assigned a session sensitivity label reflecting the maximum sensitivity of the
information they are permitted to access. Objects are assigned an object sensitivity label
reflecting the sensitivity of the data contained within. In general, objects are sensitivity labeled
with the session sensitivity label of the creating user. Containers, other than rows, which hold
MAC protected objects (e.g., databases which hold tables) may contain objects equal to or greater
than the container’s sensitivity label. **TR** databases, catalogs, schemas, tables, views, indexes,
and rows are protected containers that are MAC sensitivity labeled. A user is permitted to read an
object at a particular object sensitivity label if the session sensitivity label of the user dominates
the sensitivity label of the object. A user is permitted to write an object at a particular object
sensitivity label if the session sensitivity label of the user is equal to the sensitivity label of the
object.

**TR** provides mandatory access control features that are built atop the mandatory access control
primitives of the TOS. The security lattice defined for the TOS is used for the Trusted
Rubix protected objects and users.

The **MAC** policy enforcement in **TR** follows the six rules below:

1. A user may read a piece of information if the user’s session sensitivity label dominates
   the object sensitivity label of the information.
2. If a user creates a piece of information, that information is sensitivity labeled with the
   session sensitivity label of the user.
3. A user can modify a row of a table if its session sensitivity label equals the object
   sensitivity label of the information. There are no provisions in **TR** for an untrusted
   user to write a piece of information to a higher object sensitivity label than the user’s
   session sensitivity label (a write-up).
4. When two pieces of information are combined, the object sensitivity label of the resultant information must be chosen to dominate the object sensitivity labels of the original pieces of information. This is accomplished by taking the higher of the two hierarchical classifications and combining both sets of categories.

5. When a piece of information is extracted from another piece of information, it inherits a sensitivity label that dominates the label of the original piece of information.

6. Containers holding other MAC protected objects (all containers except rows) may contain objects with sensitivity labels equal to or greater than the container’s sensitivity label.

All objects in TR are immediately and automatically protected by MAC at creation time. For example, when a database is created by executing the CREATE DATABASE command, it is immediately assigned an object sensitivity label which matches the user's session sensitivity label. As another example, if the user inserts a row into a table, that row is assigned the user’s session sensitivity label.

There may be times when users wish to access data at a sensitivity label higher than the TOS session sensitivity label with which they connected to the database. TR provides facilities to dynamically alter the TR session sensitivity label. The user must have one of the rubix.mac.setsess.* authorizations to perform this operation.

NOTE

In addition to the Multilevel Security (MLS) Mandatory Access Control (MAC) policy, TR supports the Type Enforcement (TE) MAC policy of SELinux and a proprietary Attribute Based Accessed Control (ABAC) MAC policy of the Security Policy Manager (SPM). In general, all configured MAC policies must permit an operation for it to succeed. For more information on TE and SELinux please see the Trusted RUBIX SELinux Guide and for more information on ABAC and the SPM please see the Trusted RUBIX Security Policy Manager Reference Guide and Tutorial.

4.4.1 Row Polyinstantiation

TR associates each database object on the system with a sensitivity label. This sensitivity label is stored in the 'ROWLABEL' hidden column associated with each row. When the object is created, the sensitivity label of the object is set to the sensitivity label of the subject; i.e., the login sensitivity label of the user who created the object.

There are three distinct methods of polyinstantiation that a relational RDBMS may support (counting no support as one). The first, called POLYLOW, consists of replicating a HIGH key at LOW when a LOW subject attempts to insert a row whose key matches that of an existent HIGH row. This action is taken to close the key collision covert channel. The second method, called POLYHIGH, entails reintroduction of LOW keys at HIGH by a HIGH subject who, although fully aware of the existence of the LOW row, chooses to re-instantiate the key anyway because that is considered a semantically meaningful action. The last is to not support polyinstantiation.

It is not possible to have POLYHIGH behavior without POLYLOW. Furthermore, once a table has been created and the requisite polyinstantiation discipline declared, it is not possible to change
that discipline for the remainder of the table's lifetime.

If POLYLOW is specified with respect to a given table, then a subject may insert a row into that table if and only if no rows with that key that are dominated by the subject's sensitivity label exists in the table. In contrast, if POLYHIGH is specified, then a subject may insert a row into that table only if no row with that key exists in the table at the subject's security sensitivity label. If a row with the same key exists at a higher (or non-comparable) user sensitivity label, then the row is polyinstantiated.

If POLYNONE is specified, then the attempted creation of a record with the same key as that of an existing record, will fail.

The consequence of inserting multiple key instances into a polyinstantiated table is that, when data is read from the table, there may be various competing instances of a given row's key from which the RDBMS must choose. The record with the highest sensitivity label that is dominated is read. In addition, you may use the VIEW BY POLYINSTANTIATION construct to cause SELECTs from polyinstantiated tables to return all rows that are dominated by the reader's sensitivity label.

The PRIMARY KEY constraint is variously affected by the multilevel security architecture depending on the POLYINSTANTIATION method that a user selects for a table. A UNIQUE constraint in Trusted Rubix is enforced to be UNIQUE within a sensitivity label. In other words, a set of columns which are defined to be UNIQUE will be UNIQUE at the sensitivity label. Using a check constraint on the ROWLABEL field it is possible for to limit the sensitivity labels from which data can be added to a particular table.

**NOTE**

Multilevel security affects foreign keys in much the same manner as it affects UNIQUE constraints. The data referenced by the foreign key must exist at the same sensitivity label as the data being inserted. In other words, the combination of every foreign key value and its sensitivity label must match the primary key at the same sensitivity label. All affected tables must have the same POLYINSTANTIATION policy in effect. See the SFUG for details on this feature.

### 4.5 Security Audit

**TR** provides an extensive audit mechanism. This includes auditing of all significant events including all security related actions initiated by the user. The general user has no control over defining or manipulating audit events. The user who possesses the Audit Administrator authorizations exclusively deals with audit mechanisms and takes the necessary actions when required. General audit information is found in Chapter 7.

The audit operations are discussed under Audit Administrator commands `rxauditrpt` and `rxauditset` (see the Administrative Commands Reference Guide for more information). These operations are used to maintain the **TR** audit subsystem and to view the data collected by it.

**TR** also is flexible in its auditing policy. **TR** supports separate administrative authorizations for the management of auditing privileges. The audit authorizations confer upon the grantee the privilege to perform functions that include generating audit reports, setting audit criteria, and listing, deleting, and setting audit log files. The information on the audit authorizations can be found in Section 5.1.3.
Whenever the mandatory or discretionary security policy is invoked, the event is logged in the audit trail. Also, the system audits any action that can potentially cause access to, generation of, or effect the release of classified or sensitive information. The audit data is selectively acquired based on the auditing needs of the installation and/or application. The audit data is captured with enough granularities to trace the events to a specific individual. The audit events are tabularized in Chapter 7.

**Trusted Rubix** enables the administrative user to track or account for user actions, determine a user’s data access, detect suspicious events when they happen, and take corrective action such as removing his privilege and privileges and reversing all modifications performed by the user.

### 4.6 Trusted Recovery

**TR** handles three common failure conditions that a RDBMS may encounter and recovers it back to a consistent and secure state after a failure.

- A transaction failure;
- An inconsistent state of main memory (system failure); and
- A primary disk error (media failure).

#### 4.6.1 Write-Ahead Logging

In **TR**, write-ahead logging is used to record operations that have been submitted to the database. The logs are used to restore the database to a consistent state after one of the previously mentioned failure conditions occurs.

**TR** has two types of logs:

- the rollback/recovery log and
- the restore redo log.

The rollback/recovery log is used to recover from transaction and system failures. One such log is maintained for each database and the files corresponding to this log are stored in the database directory. The restore redo log is maintained in an administrator-defined location and is used to recover from a media failure. The default location is also in the database directory. The state of each log is placed on persistent storage at proper times in order to guarantee the database can be placed back in a correct state following a failure.

#### 4.6.2 Transaction Failure and Recovery

A transaction failure is due to a partially executed transaction performing an operation that violates the notion of correctness (i.e., serializability) defined for transactions. Examples of this in the **TR** system are read/write conflicts. When this condition is detected, the effects of the transaction are removed from the database and the transaction is marked as aborted. This is known as a "transaction rollback." The need for transaction rollback is detected by **TR** and the rollback is performed automatically. Transaction rollback may also be initiated by the user.

Transaction recovery is performed on-line automatically by the **TR** system as transaction failures occur. Rollback is executed by reading back through the rollback/recovery log and removing the effects of all changes the transaction made to the database. Sufficient disk space
must be available for maintenance of the rollback/recovery log.

4.6.3 System Failure and Recovery

The second failure condition is an inconsistent state of main memory due to a system failure. This can result from an abnormal shutdown of the system or a programming error that cannot be recovered from using the state of main memory. In this case, the system is recovered using the state of the primary disk(s). The need for system recovery is detected by Trusted Rubix and is performed automatically. Typically, system recovery should not seriously delay the execution of transactions.

System recovery is performed automatically on a per-database basis as the database is initially opened. During the recovery process, no transactions may execute. The recovery process takes the current state of the database on persistent storage along with the rollback/recovery log and produces a correct database state reflecting the updates of all transactions that committed prior to the system failure.

**WARNING**

Sufficient disk space must be available for maintenance of the rollback/recovery log.

4.6.4 Media Failure and Restore

The third failure condition is a primary disk error (media failure). Examples of this are a physically damaged hard drive or an abnormal shutdown that results in a partial disk write. In this case, the system is generally recovered using the state of a backup storage device such as a non-primary disk. This is known as "media restore."

Media restore is the responsibility of the TR Database Administrator (DBA) and can delay transaction processing. Prior to a media failure, an image of the database is dumped to a non-primary disk by the Operator. If a media failure occurs, the media is first checked for fatal defects and, if found, replaced. The previously dumped database image is then restored and the contents of the restore log are added by the TR DBA.

4.6.5 Backup

Database backup is performed using the rxdump command and is the responsibility of the TR Operator (OP). It is important to consider accidental data loss and/or loss of data integrity, while installing any computer system. TR comes with a set of utilities to recover from catastrophic loss. TR provides Operators flexibility in implementing their backup paradigm. An Operator can fully backup databases at selected intervals. An Operator can also log all database transactions between full system backups to disk. In TR, Operators are assigned the task of performing a full backup.

It is also advisable to archive each backup to keep a historical record of the state of the database(s). The archive produced can be kept as a permanent record, and can be used at a later date should the need arise. The full backup schedule that is selected is determined by database activity. If the database in question is very dynamic, then backups should be more frequent than for an equivalent relatively static database. If a dump is archived, the storage directory (named by the dump name), all contained dump files, and any relevant restore redo log files should be included in the archive.
This command should be run at SYS_HIGH MAC label to ensure that all records are read. A warning message will be printed if the command is run at a label lower than SYS_HIGH. If a lower label is used, then only dominated objects will be included in the backup. Note that this may create errors during a restore that applies redo logs as updates to higher level objects will fail because the target objects do not exist in the restored database.

### 4.6.6 Restore

The restoration of data is a two-fold process in which the backup is restored and then, overlaid with the transaction logging entries from the restore redo logs. This is performed using the `rxrestore` command and only the **Trusted Rubix** Database Administrator (DBA) is allowed to perform this operation.

**WARNING**

| The database must remain off-line until the entire restoration process is completed. The entire roll forward operation must be completed before any new transactions can be applied. |

The roll forward is optionally part of the restore process. If specified by DBA during the `rxrestore` invocation, after the database has been restored from the dump, the restore redo log directory is searched for log files. Each log file is applied to the database. When all the log files have been applied, the `rxrestore` command ends and the database is restored and ready to use.

A restored database must have a name that did not exist prior to the `rxrestore` invocation. Therefore, it will typically not have the same name as the database from which the dump was created. Once the new restored database has been created it should be tested for correct operation. Then, the `rxdb` command may be used to remove or rename the original database, if it exists, and rename the new database to the original database name.

**WARNING**

| If possible, keep the original database until the restored database has been verified to function properly. |

### 4.7 Assurance

To meet high assurance database requirements, **TR** has integrated its security mechanisms with the security mechanisms of the TOS. Only the unique combined security functionality provided by a trusted operating system and a trusted RDBMS can yield true high assurance application development and tightly controlled assignment of security attributes. Together, the TOS and **TR** provide security functionality and strong access controls. The mandatory security mechanisms of the TOS ensure that subsystems cannot be bypassed, provides a means of ensuring the end-user that he or she is interacting with trusted software, and guarantees a mutually authenticated channel. Such mechanisms may also be used to confine an application to a unique security domain that is strongly separated from other domains in the system. Applications may still misbehave, but the resulting damage can be restricted to a single security domain. This ability to confine security breaches is critical to controlling data flows in support of a system security
policy.

Other TR assurance features that are related to security mechanisms and that provide significant security capabilities are:

→ Transaction integrity, concurrency and integrity constraints, to assure the consistency and integrity of data held in a database.

→ Secure import and export of data, into the same or different database, at the same or a different sensitivity label, while maintaining data integrity and confidentiality. (See Chapter 6 for a complete description.)

→ Backup and recovery of TR databases, using specific separate administrative utilities. (see rxdump and rxrestore in the Administrative Commands Reference Guide.)

→ Three levels of row polyninstantiation (see Section 4.4.1).

Each of the following subsections provides a list of the names of the TOE evaluation evidence as security assurance measures for each EAL4 class. It also lists the security assurance requirements that are met, a description of the assurance evidence, and the rationale demonstrating these evidences meet the assurance requirements.

### 4.7.1 Configuration Management

The Configuration Management (CM) system used for TRusted Rubix is the automated UNIX configuration management system known as the Revision Control System (RCS) which tracks version changes of TR product and TOE documentation. This CM system provides the capability to restrict file access to authorized individuals, which mitigates the potential for any unauthorized changes to be introduced in the system. The Configuration Management Document provided for TR is the TR Configuration Management Manual. This manual includes the configuration management plan and procedures for using the CM system, including procedures for checking source file revisions in and out, compiling the TOE, and examining the history of file revisions. The source representation of TR is managed by RCS and is described in the CM Manual. The CM manual also describes the procedures necessary to automatically generate a binary representation of the TOE using the UNIX tools "imake", "make", and "makedepend". Together, these tools provide the capability to automatically detect changes in source representation files when generating the binary representation of the TOE.

The CM Manual provides a Configuration Item (CI) list of uniquely identifiable items that comprise the release of TR and documents CM and acceptance plans. The CI list uniquely identifies each of the following types of files:

→ TOE source files, → "make" files, and
→ API include files, → online documentation files.

The method for establishing the unique identification for each CI is documented in the CM Manual. RCS provides security features for ensuring that only authorized changes are made to configuration items and supports the generation of the TOE. The CM Manual contains an acceptance plan that describes the procedures used to accept modified or newly created configuration items as part of the TOE.

The CM Manual describes the methods and procedures used to track the TOE implementation source files and TOE documentation including:
design documents, guidance documents, and
test documents, CM documents, security flaws.

Security flaws are maintained in a web-based tool, which records needed changes in sufficient detail to coordinate source code modification kept in the RCS tool.

### 4.7.2 Fully Defined External Interfaces

The **TR** Functional Specification document describes all of the external interfaces, which include the functional application programming interfaces (API) and the administrative trusted programs and command. It provides special details for security-relevant functions. TOE Security Functions are identified and completely described while non_TSF interfaces are described and clearly indicated that they are not security-relevant functions. This document specifies all external interfaces and provides complete details of all effects, exceptions, and error messages. Since the document contains a description of all external interfaces, including security functions, it consequently contains a complete representation of the TSF.

### 4.7.3 Security Enforcing High Level Design

The **Trusted Rubix** High Level Design document describes the structure of the TSF in terms of the following subsystems:

- Server Interface,
- SQL Engine,
- Kernel, and
- Common Server.

All interfaces to each subsystem are described individually and identified as TOE internal or external and either TSP-enforcing or non-TSF enforcing interfaces. It also describes the security functionality provided by each subsystem of the TSF, such as the mandatory access control (MAC) and discretionary access control (DAC) security functions. This design document also identifies the underlying TOS MAC and DAC functions and their relationship to **TR**.

### 4.7.4 Security Features User Guide (SFUG)

The **Security Features User Guide** (SFUG) details the functions and interfaces available to non-administrative users. The guidance includes description, syntax, options explanation, and example usage for each interface. Warnings associated with each security interface are offset in boxes to increase visibility. The **SQL Reference Guide** details the full syntax of all commands accessible by users, including all security mechanisms. The SFUG presents all user responsibilities for secure operation of the TOE, including assumptions regarding user behavior. For example, the SFUG includes the assumption that most users will authenticate at the highest privilege permitted and record data at that level, even when the information does not require that level of protection. In addition to describing the TOE security requirements, the SFUG describes all relevant security requirements for the IT environment. Specifically, the SFUG describes the dependency of **TR** on the TOS for database security labels.

### 4.7.5 Life Cycle Support

The **TR** Life Cycle Support document describes the software development model, security measures, and development tools used to construct the TOE. The processes and procedures
described in the document provide the assurance that the software is developed and maintained without adverse impact to TOE security functions. The document provides a description of the physical and procedural countermeasures used at ITI to protect the TOE during development.

The traditional "Waterfall Model" is the life-cycle model used to develop TR. The document illustrates each stage in the model and describes the activities ITI performs during each stage. For example, the Engineering Change Request (ECR) process describes the actions required to incorporate new functionality into TR. Specifically, the process describes the individual roles and responsibilities in the ECR process as well as the tools used to track the request.

Software development tools provide additional assurance that TOE Security Functions operate as specified. ITI uses several tools for development and verification of software developed in the "C" language. For example, ITI uses tools such as "Valgrind" and "Insure++" to verify correct memory usage and detect memory overruns and uninitialized variables. Additionally, the UNIX debugging tool "gdb" is used to trace software during runtime in an effort to locate defects and assure correct execution.

### 4.8 Security-relevant Events

There are three security relevant events that the administrators of TR must be prepared to address. These are an overflow of the audit logs, a crash of the TR server, and updates to the user base.

Management of the audit logs is discussed in Section 7.4.4.

Be prepared to restore the database in case of a system crash. The key to addressing a crash of the TRUSTED RUBIX server is to have a current backup of the database to perform a restoration from. Backing up the database is discussed in Section 6.3. Restoring the database is discussed in Section 6.2.

Manage DAC privileges. Updates of the user base must be coordinated with the TOS administrator. Since TR's MAC policy and authorizations are fully integrated with the TOS, there is no need for database administration of these mechanisms. However, the DAC security mechanism of TR requires maintenance actions by the Database Administrator in case of a user’s access privilege being revoked. User management is discussed in Section 4.1, and Section 4.1.3 describes how to list the DAC privileges held by the user. Once an expired user’s DAC privileges are known, the administrator must decide whether it is appropriate revoke (or NULL) the privileges, or grant them to another user.

### 4.9 Documentation

Just as important as the assurance that TR operates correctly, is the documentation which describes the operation of the trusted facility management tools and the effects of their use. Chapter 5 of this manual informs administrative users of the identity and use (and potential misuse) of the various administrative roles and the operations performed within their purview. Chapter 6 describes how the DBA ensures the secure operation of TR. Chapter 7 describes audit administration tools and procedures, and Chapter 9 describes how to establish TR initial configuration.

In addition to being familiar with the Security Features User's Guide (SFUG), the SQL-ODBC
Tutorial and the SQL Reference Guide, the administrator should also have available and be familiar with the TOS Trusted Facility Manual (TFM), and the TOS SFUG.
CHAPTER 5  TRUSTED RUBIX AUTHORIZATIONS AND ROLES

**Trusted Rubix** exploits the RBAC mechanism provided by the TOS to give users the ability to execute special security operations or supersede the MAC and/or DAC security policies. This provides the mechanisms by which authorized users may specify how to enforce the security policy of their organization and define administrative roles in a manner such that they support the intent of the security policy. The individual authorizations are presented in Section 5.1.

**TR** provides a set of default roles each with pre-assigned **TR** specific authorizations. The default roles are automatically created during the installation of **TR**. The default roles and their associated authorizations are given in the following table.

<table>
<thead>
<tr>
<th>Default TR Role</th>
<th>Authorizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Administrator</td>
<td>rubix.audit.*</td>
</tr>
<tr>
<td>Database Administrator</td>
<td>rubix.dac.<em>, rubix.admin.</em></td>
</tr>
<tr>
<td>Operator</td>
<td>rubix.restore.backup.<em>, rubix.restore.logs.ls</em>, rubix.restore.logs.rm.<em>, rubix.restore.logs.set.</em></td>
</tr>
<tr>
<td>Security Administrator</td>
<td>rubix.mac.<em>, rubix.</em>,grant, rubix.restore.create.*</td>
</tr>
<tr>
<td>User</td>
<td>rubix.user.*</td>
</tr>
</tbody>
</table>

The default **TR** security administrator role holds the `rubix.*.grant` authorization. A user who assumes this role is therefore free to group authorizations and create new **TR** database roles to meet security requirements. The main reason for creating a role is to define an explicit job responsibility that can use special commands and actions and hold any necessary privileges, which need to be isolated from normal users.

The default roles are discussed in Section 5.2.

### 5.1 Authorizations

Because **TR** uses the Role Based Access Control (RBAC) mechanism provided by the TOS, the full robustness of the RBAC mechanism may be utilized in one seamless environment for both **TR** and the TOS. As a result, the site administrators may create roles with any combination of authorizations regardless of whether those authorizations are for **TR**, the TOS, or any other authorization type.

The **TR** authorizations only have significance for operations on **TR** software (i.e., they imply no privilege for any other software).

**WARNING**

For an authorization to be effective it must be present on the machine that contains the database being accessed. For more information on using authorizations see the TOS documentation.

All of the **Trusted Rubix** authorizations begin with “rubix.” and then proceed with
more fine grained names. This is analogous to an internet domain address. The more general name parts are to the left of the more specific name parts. If a user is given the `rubix.*` authorization then he/she is given all of the authorizations available for TR. If he/she is given the `rubix.mac.*` authorization then he/she is given all of the TR MAC authorizations.

The TR authorizations have been created to incorporate database names into them where appropriate (Solaris based systems only). For instance, the `rubix.mac.recl.upgrade.dbname` authorization may be given to a user for only a specific database in the form `rubix.mac.recl.upgrade.dbname` or for all databases in the form `rubix.mac.recl.upgrade.*`. This allows site administrators to partition the trusted responsibilities across the physical structure of the databases. In order to use the database name in the authorization the TOS Security Administrator must add appropriate authorizations, based upon the specific database names used by the site, to the `/etc/auth_attr` file (Solaris based systems only).

**WARNING**

If no additional authorizations are added after the initial TR install then the authorizations given below may only be used without database name extensions.

The ability to assign authorizations to users is controlled with the `rubix.*.grant` authorization (Solaris based systems only).

### 5.1.1 Trusted RUBIX MAC Authorizations

The TR MAC authorizations provide the ability to change the sensitivity label of a database row and to change the database session sensitivity label to one that is different from the TOS session label at the time the TR session was initiated. Each authorization type is partitioned into changing a label “up”, changing a label “down”, and changing a label “across”. Since changing a label to an incomparable label technically falls into neither of these categories, the partitioning is done based upon raising or lowering the hierarchical classification portion of the sensitivity label. Altering the session label is further broken down into read only and read/write categories. Note that the `rubix.mac.recl.*` authorizations may be combined with the `rubix.user.import` authorization to allow multilevel import operations.

#### Upgrade the Sensitivity Label of A Row

**rubix.mac.recl.upgrade.dbname**

Use the SQL UPDATE command to change the label of a row to one where the new label dominates the row’s original label. If the user also holds the `rubix.user.import.dbname` authorization the user may use the multilevel import facility to set the sensitivity label of the database rows to one that is greater than their current TOS session label.

#### Downgrade the Sensitivity Label of A Row

**rubix.mac.recl.downgrade.dbname**

Use the SQL UPDATE command to change the label of a row to one where the new label is dominated by the row’s original label. If the user also holds the `rubix.user.import.dbname` authorization the user may use the multilevel import facility to set the sensitivity label of the...
database rows to one that is lower than their current TOS session label.

**Change the Sensitivity Label of A Row To An Incomparable**

`rubix.mac.recl.across.dbname`

Use the SQL `UPDATE` command to change the label of a row to one where the new label is incomparable with the row’s original label. If the user also holds the `rubix.user.import.dbname` authorization the user may use the multilevel import facility to set the sensitivity label of the database rows to one that is greater than their current TOS session label.

**Raise Session Label For Read Only**

`rubix.mac.setsess.uprd.dbname`

Use the `ALTER SESSION SET LABEL` command to raise the database session label above the initial logon database session sensitivity label and issue read only transactions. This is bounded by the user’s current TOS session clearance.

**Raise Session Label For Read/Write**

`rubix.mac.setsess.uprdwrt.dbname`

Use the `ALTER SESSION SET LABEL` command to raise the database session label above the initial logon database session sensitivity label and issue read/write transactions. This is bounded by the user’s current TOS session clearance.

**Across Session Label For Read Only**

`rubix.mac.setsess.acrossrd.dbname`

Use the `ALTER SESSION SET LABEL` command to change the database session label to one incomparable to the initial logon database session sensitivity label and issue read only transactions. This is bounded by the user’s current TOS session clearance.

**Across Session Label For Read/Write**

`rubix.mac.setsess.acrossrdwrt.dbname`

Use the `ALTER SESSION SET LABEL` command to change the database session label to one incomparable to the initial logon database session sensitivity label and issue read/write transactions. This is bounded by the user’s current TOS session clearance.

**Lower Session Label For Read/Write**

`rubix.mac.setsess.downrdwrt.dbname`

Use the `ALTER SESSION SET LABEL` command to lower the database session label below the initial logon database session sensitivity label and issue read/write transactions. This is bounded by the user’s current TOS session clearance.
Grant Authorization

rubix.*.grant

Allows partitioning of authorizations between various TR security roles (Solaris based systems only).

5.1.2 Trusted RUBIX DAC Authorizations

The Trusted RUBIX DAC authorizations give the privileged user the ability to perform SQL commands without regard for the DAC security policy. The authorizations are partitioned by SQL operations.

**WARNING**

SELECT ability is implied for SQL operations that also may require SELECT access to a sub-query, such as the INSERT operations, by the given authorization. For instance, the rubix.dac.rxinsert.dbname authorization will give the user the ability to perform the “INSERT INTO t1 SELECT * FROM t2” even if the user does not hold the DAC SELECT privilege on table t2.

The DAC authorizations do not supersede the DAC policy for write operations on specially protected Definition Schema objects. These objects are contained in the SYSTEM catalog and include the SYSTEM catalog itself.

DAC Select

rubix.dac.rxselect.dbname

Supersede DAC privileges when performing a SELECT command on a table or view.

DAC Insert

rubix.dac.rxinsert.dbname

Supersede DAC privileges when performing an INSERT command on a table or view. The DAC policy is not superseded for objects in the SYSTEM catalog.

DAC Update

rubix.dac.rxupdate.dbname

Supersede DAC privileges when performing an UPDATE command on a table or view. The DAC policy is not superseded for objects in the SYSTEM catalog.

DAC Delete

rubix.dac.rxdelete.dbname
Supersede DAC privileges when performing a c command on a table or view. The DAC policy is not superseded for objects in the SYSTEM catalog.

**DAC Create**

`rubix.dac.rxcreate.dbname`

Supersede DAC privileges when performing a CREATE or ALTER TABLE ADD CONSTRAINT /COLUMN command on any object. The DAC policy is not superseded for objects in the SYSTEM catalog.

**DAC Drop**

`rubix.dac.rxdrop.dbname`

Supersede DAC privileges when performing a DROP or ALTER TABLE DROP CONSTRAINT /COLUMN command on any object. The DAC policy is not superseded for objects in the SYSTEM catalog including the SYSTEM catalog itself.

**DAC Grant**

`rubix.dac.rxgrant.dbname`

Supersede DAC privileges when performing a GRANT command (see Section 3) on any object. The DAC policy is not superseded for objects in the SYSTEM catalog including the SYSTEM catalog itself.

**DAC Revoke**

`rubix.dac.rxrevoke.dbname`

Supersede DAC privileges when performing a REVOKE command on any object. The DAC policy is not superseded for objects in the SYSTEM catalog including the SYSTEM catalog itself.

### 5.1.3 Trusted RUBIX Audit Authorizations

The Trusted RUBIX audit authorizations provide the ability to perform functions related to the TR audit mechanism. This includes generating audit reports, setting audit criteria, and listing, deleting, and setting audit log files.

**Audit Report**

`rubix.audit.report.dbname`

Use the `rxaudit rpt` command to produce and view an audit report.

**Audit Set Criteria**

`rubix.audit.setcrit.dbname`

Use the `rxaudit set` command to set the audit criteria.
List Audit Log Files
rubix.audit.logs.ls.dbname

Use the rxlogs command to list the audit log files.

Delete Audit Log Files
rubix.audit.logs.rm.dbname

Use the rxlogs command to delete audit logs.

Set Audit Log Directory/Files
rubix.audit.logs.set.dbname

Use the rxlogs command to specify the current directory to hold the audit log files.

5.1.4 Trusted RUBIX Restore Authorizations

The TR restore authorizations provide the ability to perform functions related to the TR dump/restore mechanism. This includes producing backups of databases, performing database restores, and listing, deleting, and setting restore log files.

List Restore Log Files
rubix.restore.logs.ls.dbname

Use the rxlogs command to list the restore log files.

Delete Restore Log Files
rubix.restore.logs.rm.dbname

Use the rxlogs command to delete the restore log files.

Set Restore Log Directory/Files
rubix.restore.logs.set.dbname

Use the rxlogs command to specify the current directory to hold the restore log files.

Backup Database
rubix.restore.backup.dbname

Use the rxdump command to perform a database backup.

Restore Database
rubix.restore.create.dbname
Use the `rxrestore` command to restore a database.

### 5.1.5 Trusted RUBIX Administrative Authorizations

The **Trusted Rubix** administrative authorizations provide the ability to perform functions related to general administration.

**Drop Database**

```
rubix.admin.db.rm.dbname
```

Use the `rxdb` command to drop a database.

**List Databases**

```
rubix.admin.db.ls.dbname
```

Use the `rxdb` command to list the current databases.

**Move (Rename) Databases**

```
rubix.admin.db.mv.dbname
```

Use the `rxdb` command to move (rename) the current databases.

**Dispatcher Startup**

```
rubix.admin.dspr.start
```

Use the `rsrvrman` command to start the **TR** Dispatcher.

**Dispatcher Termination**

```
rubix.admin.dspr.term
```

Use the `rsrvrman` command to terminate the **TR** Dispatcher.

**Server Termination**

```
rubix.admin.svr.term
```

Use the `rsrvrman` command to terminate a **TR** Server process.

### 5.1.6 Trusted RUBIX User Authorizations

The **TR** user authorizations provide the ability to perform functions given to typical users. Currently this includes the ability to import data into and export data out of a database. If the user holds any `rubix.mac.recl.*` authorization in addition to the `rubix.user.import.dbname` authorization the user may perform a multilevel import, limited by his/her session sensitivity label.
Export Data

rubix.user.export.dbname

Use the rxexport command to export data from a table or view. If the user also holds the rubix.dac.rxselect.dbname authorization the DAC policy will be superseded.

Import Data

rubix.user.import.dbname: Use the rximport command to import data. If the user also holds the rubix.mac.recl.upgrade.dbname the user may use the multilevel import facility to set the sensitivity label of the database rows to one that is greater than their current TOS session label. If the user also holds the rubix.mac.recl.across.dbname the user may use the multilevel import facility to set the sensitivity label of the database rows to one that is incomparable with their current TOS session label. If the user also holds the rubix.mac.recl.downgrade.dbname the user may use the multilevel import facility to set the sensitivity label of the database rows to one that is less than their current TOS session label. If the user also holds the rubix.dac.rxinsert.dbname authorization the DAC policy will be superseded.

5.2 Trusted RUBIX Default Roles

5.2.1 Audit Administrator

The Trusted RUBIX default Audit Administrator (AUD) role is responsible for administering the TR audit subsystem. This role is assigned the following authorizations at the time of TR installation:

rubix.audit.report.dbname:
rubix.audit.setcrit.dbname:
rubix.audit.logs.ls.dbname:
rubix.audit.logs.rm.dbname:
rubix.audit.logs.set.dbname:

Please note that there is no requirement to group the audit authorizations all under the default role, and the user is free, and encouraged, to create more limited roles by grouping the audit authorizations to meet their security requirements (Solaris based systems only).

WARNING

Since an auditor has the capability to disable the recording of actions which he performs, it is suggested that (given sufficient qualified staff), users capable of operating in this role not be allowed to operate in other administrative roles.

For details of the rxlogs, rxauditset and rxauditrpt commands see the Administrative Commands Reference Guide.

5.2.2 Database Administrator

The Trusted RUBIX default Database Administrator (DBA) role is empowered to perform all operations which maintain the consistency and integrity of the stored data. The
Database Administrator is assigned the following authorizations at the time of TR installation:

- `rubix.dac.rxcreate.dbname`
- `rubix.dac.rxdelete.dbname`
- `rubix.dac.rxdrop.dbname`
- `rubix.dac.rxgrant.dbname`
- `rubix.dac.rxinsert.dbname`
- `rubix.dac.rxrevoke.dbname`
- `rubix.dac.rxselect.dbname`
- `rubix.dac.rxupdate.dbname`
- `rubix.admin.db.ls.dbname`
- `rubix.admin.db.mv.dbname`
- `rubix.admin.db.rm.dbname`
- `rubix.admin.dspr.start`
- `rubix.admin.dspr.term`
- `rubix.admin.svr.term`

The authorizations granted to the default DBA administrative role are largely DAC authorizations which make him/her DAC exempt, but not MAC exempt. For example, to alter the Access Control Lists (ACLs) on a table, the DBA must be operating at the table's label. If the DBA's label strictly dominates the table's label then the table's ACL cannot be altered. If the table's label strictly dominates the DBA's label, then the table will not be visible to the DBA.

The exception to this generality is the admin authorization which permits usage of the `rxdb` command which carries MAC privilege to manipulate data.

For details of the TRusted RubIX commands see the Administrative Commands Reference Guide. For more information on the DAC security policy see Chapter 2 of the Security Features User's Guide.

### 5.2.3 Operator

The TR default Operator (OP) role is authorized to perform functions related to the TR dump/restore mechanism. This includes producing backups of databases, performing database restores, and listing, removing, and modifying restore log files. This role is assigned the following authorizations when TR is installed:

- `rubix.restore.logs.ls.dbname`
- `rubix.restore.logs.rm.dbname`
- `rubix.restore.logs.set.dbname`
- `rubix.restore.backup.dbname`

No other responsibilities are associated with this role. Although operators are able to back-up and restore entire databases, they hold no other special capability as far as reading or writing those databases, (i.e.: SELECT or UPDATE) but are, in this respect, bound by the standard MAC and DAC mechanisms. For details of the `rxlogs` and `rxdump` command see the Administrative Commands Reference Guide.

### 5.2.4 Security Administrator

The TRusted RubIX default Security Administrator (SA) role is responsible for all operations which may arbitrarily determine the label of a database object. This role is assigned the following authorizations when TR is installed:

- `rubix.mac.recl.upgrade.dbname`
- `rubix.mac.recl.downgrade.dbname`
- `rubix.mac.across.dbname`
- `rubix.mac.setsess.downrdwrt.dbname`
- `rubix.mac.setsess.acrossrd.dbname`
- `rubix.mac.setsess.acrossrdwrt.dbname`
- `rubix.*.grant`
- `rubix.restore.create.dbname`
The TR MAC authorizations provide the ability to change the sensitivity label of a database row and to change the database session sensitivity label to one that is different from the TOS session label at the time the TR session was initiated. Each authorization type is partitioned into changing a label “up”, changing a label “down”, and changing a label “across” (i.e., to an incomparable label). Altering the session label is further broken down into read only and read/write categories. Note that the rubix.mac.recl.* authorizations may be combined with the rubix.user.import authorization to allow multilevel import operations.

For details of the rxrestore and rximport, commands see the Administrative Commands Reference Guide.

Altering Session Labels

The TOS label (OSLABEL) of a user is the label that is assigned to that user by the underlying operating system at login time. When a user connects to TR, the label assigned is the session label. By default the system label and session label start out equal. The session label is used to classify all objects inserted into any database. It is also used to enforce all MAC policies. In effect, TR clears users within the database for the range of labels for which the user has been cleared by the TOS.

There may be times when users wish to access data at a label higher than the label at which they are connected to the database. TR provides facilities to dynamically alter the session label. The user must possess the rubix.mac.setsess.* authorizations to perform this operation. In no case may the session label strictly dominate the user’s current TOS clearance.

The following example is given to illustrate the effect of altering the session label upon a SQL SELECT. In this example, a table named projects is assumed to contain project numbers (pno) classified with an ascending linear ordering of four sensitivity labels (rowlabel):

- UNCLASSIFIED
- CONFIDENTIAL
- SECRET
- TOP_SECRET

A user is assumed who will access the table at two different levels of access privilege, labeled: SECRET and TOP_SECRET. The table itself carries a classification (sensitivity label) of CONFIDENTIAL.

The user first accesses the table with an access privilege (sensitivity label) of SECRET, and the only rows that will be visible to the user are the rows dominated by his/her access privilege.

```
SELECT rowlabel, pno
FROM projects;
```

<table>
<thead>
<tr>
<th>rowlabel</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNCLASSIFIED</td>
<td>FCS</td>
</tr>
<tr>
<td>SECRET</td>
<td>MGS</td>
</tr>
</tbody>
</table>
Note that there is no row in the view with a sensitivity label of ‘TOP_SECRET’. Now assume the user possesses `rubix.mac.setsess.uprd.*` authorization and a TOS clearance of TOP_SECRET, and is therefore permitted to alter his/her session label to TOP_SECRET, by entering the command:

```
ALTER SESSION SET LABEL = 'TOP_SECRET';
```

In addition to returning a view of the rows shown above, the SELECT command will now also return the rows labeled at TOP_SECRET.

```
SELECT rowlabel, pno
FROM projects;
```

Note that now there is a row with a label of ‘TOP_SECRET’ in the view. This is because the user modified their session label to TOP_SECRET and their session sensitivity label now dominates the sensitivity label of the record that is classified TOP_SECRET.

All normal operations performed on any database object will proceed as if the user had logged in at the new session label. For example, an INSERT will be labeled at the altered session label, not the original session/system label. All the normal security semantics apply.

**NOTE**

In addition, no pending transactions may be active when altering one’s session label. To terminate a transaction you must either ROLLBACK or COMMIT the existing transaction.

To reset your session label to your system label, enter the following command:

```
ALTER SESSION SET LABEL = OSLABEL;
```

**NOTE**

Altering the session label does not alter the system label at which the user logged in.
Granting Authorizations

The Security Administrator (holding the \textit{rubix.*.grant} authorization) may allocate \textit{TRUSTED RUBIX} authorizations to users in any way that satisfies their particular security requirements (Solaris based systems only). This gives the Security Administrator the flexibility to partition the trusted responsibilities between any number of \textit{TR} security roles.

5.2.5 User Authorizations

The following authorizations are assignable to any \textit{TR} user:

\textit{rubix.user.export.dbname}

\textit{rubix.user.import.dbname}

5.3 Operations Associated with Default Trusted RUBIX Roles

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Produce Audit Report</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Set Audit Criteria</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>List Audit Log Files</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Delete Audit Log Files</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Set Audit Log Directory/Files</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DAC Exemption CREATE, DELETE, DROP, GRANT, INSERT, REVOKE, SELECT, UPDATE</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Database Restore (create)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Drop Database</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>List Current Database</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Move Current Database</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Database Backup</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>List Restore Log Files</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Delete Restore Log Files</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Set Restore Log Directory</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Alter Session Set Label (Lower)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Alter Session Set Label (Raise)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Alter Session Set Label (Raise - Read Only)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Upgrade Row Label</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Downgrade Row Label</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Grant Authorizations</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Multilevel Data Import</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>
5.4 Overview of Solaris Role Based Access Control

This section only applies to Trusted RUBIX hosted by the Trusted Solaris 8 or Solaris 10 TX operating systems. For information on SELinux RBAC, see the Trusted RUBIX SELinux Guide.

The Solaris operating systems provides a general purpose Role Based Access Control Mechanism which may be utilized by applications. Fundamental to the concept of Role Based Access Control for Solaris are authorizations, execution attributes, and rights profiles.

An authorization is a right assigned to users that is checked by certain privileged programs to determine whether users can execute controlled functionality. The authorization mechanism allows authorizations to be defined by the application developers and checked at runtime by the application program. Trusted Rubix uses the authorization mechanism to control which users may perform administrative functions and supersede the security policies.

An execution attribute is a set of sub-attributes associated with a specific executable program. These attributes define the effective User ID and Group ID and privilege set that the executable program’s process acquires at runtime. The execution profile provides similar functionality to the set-uid and set-gid bits on UNIX executable programs. The difference is that execution attributes are given to specific users while the UNIX execution bits apply to any user who executes the program. This restriction of functionality provides a higher degree of security. TR uses the execution attribute mechanism to control who may successfully execute TR programs.

Authorizations and execution attributes are generally bundled together into packages called rights profiles. Each rights profile corresponds to a particular job responsibility, or role. These roles may be a typical user or a trusted administrator. There are predefined rights profiles that correspond to Solaris functionality and TR functionality. A user may have multiple rights profiles associated with his/her account. TR uses the Solaris rights profile mechanism to implement TR Roles.

Solaris provides two types of login accounts:

→ the basic user account and
→ the role account.

The basic user account corresponds to a typical UNIX user account. The role account is typically used by users operating with special trusted abilities. Solaris enforces additional security restrictions on role accounts above and beyond those enforced on user accounts. For instance, a
user may generally logon to a role account only from the console. Role accounts are associated with user accounts and a user must first log in to their user account and then login to a role account. Rights profiles may be assigned to both user accounts and role accounts.

In order for a rights profile to be effective for an account, the account must be created using one of the special profile execution shells. The shells that interpret profiles are pfcs, pfksh, and pfsh. These are used when the account is created and correspond to the traditional UNIX csh, ksh, and sh shells.

Confusion may occur between Solaris role accounts and general Trusted Rubix administrative roles. The former is a specific type of physical login account and is purely a Solaris concept. The latter is a concept that consists of logical groupings of abilities to perform trusted functionality. The Solaris role account may or may not be used to implement an administrative role. For the remainder of this document unless the term “role account” is specifically used the TR administrative role is meant.

5.4.1 Manipulation of Authorizations, Execution Attributes, and Rights Profiles

Solaris maintains information about authorizations, execution attributes, rights profiles in the /etc/security/auth_attr, /etc/security/exec_attr, and /etc/security/prof_attr files respectively. It maintains information about rights profiles assigned to a user or role account in the /etc/usr_attr file. Each of these may be used with other data sources such as the NIS map and the NIS+ table.

These security database files may be edited directly to manipulate their associated security information. However, with the exception of the /etc/security/auth_attr file, this is generally discouraged. A more robust method is to use the Solaris Management Console.

5.4.2 Solaris Security Database Files

The set of authorizations recognized by the system are stored in the /etc/security/auth_attr file. When TR is installed a default set of TR authorizations is placed into this file. An example of the entries for the default TR MAC authorizations follows:

\[
\begin{align*}
\text{rubix.mac} & : : \text{:::Trusted RUBIX MAC Authorizations::help = mac.html} \\
\text{rubix.mac.recl} & : : \text{:::Trusted RUBIX MAC Reclassify Row Sensitivity Label Authorizations::help = macrecl.html} \\
\text{rubix.mac.upgrade} & : : \text{:::Trusted RUBIX MAC Upgrade Row Sensitivity Label Authorizations::help = macreclup.html} \\
\text{rubix.mac.downgrade} & : : \text{:::Trusted RUBIX MAC Downgrade Row Sensitivity Label Authorizations::help = macreclldn.html} \\
\text{rubix.setsess} & : : \text{:::Trusted RUBIX MAC Set Session Label Authorizations::help = macsess.html} \\
\text{rubix.setsess.uprd} & : : \text{:::Trusted RUBIX MAC Raise Session Label Read Only Authorizations::help = macsessuprd.html}
\end{align*}
\]
rubix.mac.setsess.uprdwrt:::Trusted RUBIX MAC Raise Session Label Read/Write
Authorizations::help = macesssuprdwrt.html

rubix.mac.setsess.downrdwrt:::Trusted RUBIX MAC Lower Session Label Read/Write
Authorizations::help = macesssndrdwrt.html

Each entry contains an authorization followed by an authorization name and an optional
associated help file. Each authorization has a multi-part name which reflects its logical grouping.
For instance, examining the rubix.mac.setsess.downrdwrt authorization reveals that it is part of
the Trusted RUBIX authorization group. It is further part of the MAC authorization
subset. This multi-part naming allows the assignment of authorizations to be performed by
group. For instance, giving an account the rubix.mac.* authorization gives all authorizations
under the MAC subset.

The auth_attr file is the only security database file that may be edited directly. TR allows its
authorizations to be specified for specific databases. This allows an account the ability to perform
the associated operation only for that database. Because the names of databases were not known
before hand these authorizations were not created during the installation of TR. To use database
specific authorizations a new authorization should be added to the auth_attr file with the database
name appended on the end. For instance to create a rubix.mac.setsess.downrdwrt authorization for
a database named “MYDB” an entry should be added to the auth_attr file as follows:

rubix.mac.setsess.downrdwrt.MYDB:::Trusted RUBIX MAC Lower Session Label Read/Write
Authorization for MYDB::help=macsessdnrdwrtmydb.html

Once the entry is added to the auth_attr file it will appear in the GUI Solaris Management Console
and may then be assigned to a rights profiles. Database specific authorizations may only be used
for fully resolved authorization names. Therefore the authorization rubix.mac.MYDB is not valid
and would be ignored by TR programs.

The exec_attr file stores attributes associated with an executable upon runtime. These attributes
include the effective User ID, the effective Group ID, and the Trusted Solaris 8 privilege set. Each
entry in the exec_attr file corresponds to exactly one rights profile. An example of an entry in this
file follows:

Rubix Security Administrator :tsol:cmd:::/opt/RUBIXserver/pub/bin/rximport:euid=rubix

This entry states that when an account with the TR Security Administrator rights profile
executes the program /opt/RUBIXserver/pub/bin/rximport it will run with an effective User ID of
rubix. There should be no reason to edit this file or modify TR execution attributes in any way.

The prof_attr file stores rights profiles and their associated attributes. Each entry consists of the
rights profile, a descriptive name, an optional help file, and a set of associated attributes. An
example of an entry in this file follows:

Rubix Security Administrator:::Rubix Security Administrator:help = rxsa.html;auths =
rubix.*.grant, rubix.mac.*

This entry states that the TR Security Administrator rights profile has the rubix.*.grant
authorization and all of the rubix.mac authorizations associated with it. This file should not be
edited directly. Any rights profile creations or modifications should be performed with the GUI
Solaris Management Console.
The `user_attr` file associates rights profiles with user and role accounts. Though not recommended, this file may also be used to directly assign individual authorizations to an account. An example of an entry in this file follows:

```
rxsa::::profiles=Rubix Security Administrator, Rubix User, Convenient Authorizations, Basic Solaris User;type=normal
```

This entry assigns four rights profiles to the normal (i.e., non-role account) user `rxsa`:

- → the **Trusted Rubix Security Administrator**, **TR User**, Convenient Authorizations, and Basic Solaris User.

The first two are **TR** rights profiles and the latter two are Trusted Solaris 8 rights profiles. This file should not be edited directly. Any assignment or revocation of rights profiles should be performed with the Solaris 8 Management Console GUI.

### 5.4.3 Using the Solaris Management Console GUI

Because **TR** security administration is fully integrated with Solaris security administration, **TR** administrative role creation and management is performed via the Solaris Management Console function in the same way as the Solaris role administration. Although **TR** specific authorizations can be grouped by the user as needed to create administrative roles appropriate to the specific application, **TR** does provide default administrative roles when it is installed.

The following screen displays illustrate the steps involved in adding (creating), managing, and assigning a **TR** administrative rights profile.

The screen above shows the Solaris Security Management Console with the menu item selected that provides the interface to add (create) a rights profile.

The screen above is the interface provided for managing rights profiles. The rights profile helps to define a user’s security environment at login. In this example, the default profile “RUBIX Database Administrator” is selected. Once selected, a number of administrative actions can be taken by clicking on the buttons at the top of the windows or by selecting the pull down menus. For example, clicking the “delete” button will delete the selected rights profile, and clicking the “properties” button will open the profile for updates to be applied.

The screen above shows the rights editing screen. The bar at the top of the window identifies the rights profile being edited as “RUBIX Database Administrator.” The window on the left shows authorizations that are not included in the profile, and the window on the right shows authorizations that have been already been given to the profile. By selecting an authorization on the left and clicking the “add” button, the security administrator can grant additional authorizations to the **Trusted Rubix** DBA rights profile. By selecting an authorization on the right and clicking the “remove” button, the security administrator can revoke authorizations grouped in the **TR** DBA profile. If an authorization is shaded, then the user is not entitled to manipulate the authorization. The `rubix.*.grant` authorization must be held by the security administrator to manipulate **TR** authorizations.

The next screen illustrates how rights profiles are assigned to user and role accounts.

In the above screen the user “tstuser” has already been granted the two roles “Convenient Authorizations” and “RUBIX Database Administrator”. By selecting a rights profile and clicking
the “add” or “delete” button, the rights profile can be granted or revoked.

If an entry in the “Available Rights” or “Granted Rights” column is shaded, then the user is not entitled to grant or revoke that rights profile. For example, if the Solaris administrator does not hold the `rubix.*.grant` authorization, all `rubix.*` authorization based rights profiles would be shaded.
CHAPTER 6 DATABASE ADMINISTRATION

This chapter discusses the following tasks that need to be performed as part of the secure administration of Trusted Rubix databases:

- Bulk Loading  →  Restore
- Recovery  →  Row Reclassification
- Backup  →  Miscellaneous

6.1 Bulk Loading

Trusted Rubix provides several mechanisms to support bulk transfer of data into and out of the database. The mechanism used will differ depending upon the kind of data to be processed and the security ramifications. The two basic data bulk loading requirements are:

- same label load
- different label load

These requirements are met by the rximport administrative utility, described in more detail in the Administrative Commands Reference Guide.

6.1.1 Same Label Load

The rximport program is used to process bulk data loads at the same level. It accepts text or binary files and imports that data into the database.

If no format file is provided, rximport reads the data file in free form mode:

1. Each record of the data file produces one row of a table.
2. Within each record, successive fields are used as the input for successive columns. If there are fewer fields on a record than there are columns, the unfilled columns are marked as Unknown.
3. Successive fields must be separated by a delimiter.

If a format file is provided, the formatted data files contain one or more input records for each row to be loaded. The format file indicates the position and length of each input column, as well as the number of lines per row. Position is a function of line and column keywords which are set to 1 initially.

WARNING

This command can only be executed on the server. To successfully use this command, the invoker must have the rubix.user.import.dbname authorization. They must also have the INSERT privilege on TABLE or have the rubix.dac.rxinsert.dbname authorization. The TOS permissions of the invoker must allow READ access to the TOS input files or the command terminates with an error message. When used in this manner, each record loaded into the database is assigned the sensitivity label of the person running the load operation.
6.1.2 Different Label Load

There are two methods of assigning labels to the imported rows:

→ Data at the same level: When `rximport` is used in this manner, the session sensitivity label must dominate the table sensitivity label and each record loaded into the database is assigned the sensitivity label of the user session running the load operation. Only single level loading is available to a non-administrative RDBMS user. To successfully use this command, the user must have `INSERT` privilege on the targeted table and `EXECUTE` privileges on the containing database, catalog, and schema or be operating within the RDBMS Database Administrator role.

→ Data at different levels: The Security Administrator can load multilevel data using the `rximport` command by specifying the `-m` flag. The `rximport` command with the “-m” option can only be used by a Security Administrator to import multilevel data into the database. Data is loaded with the row sensitivity label as the first column.

**WARNING**

The `-m` option can only be used by those with the `rubix.mac.recl.* authorization. The only constraint is that the session label of the security administrator must dominate the label of the file used as input. The `rubix.mac.recl.upgrade.dbname` is used to raise the sensitivity label of the row above that of the user’s session sensitivity label. The `rubix.mac.recl.downgrade.dbname` is used to lower the sensitivity label of a row below the user’s session sensitivity label. The `rubix.mac.recl.across.dbname` authorization is used where the new label is incomparable with the row’s original label. These security requirements are in addition to those needed to perform a “Same Level Load.”

6.2 Recovery

6.2.1 Background

In general, recovery brings the system back to a correct state after a failure. There are three common failure conditions that a database system must address.

The first is a transaction failure. A partially executed transaction performs an operation that violates the notion of correctness (i.e., serializability) defined for transactions. Examples of this in the `Trusted Rubix` system are read/write conflicts. When this condition is detected, the effects of the transaction are removed from the database and the transaction is marked as aborted. This is known as a "transaction rollback." The need for transaction rollback is detected by the database code and the rollback is performed automatically.

The second failure condition is an inconsistent state of main memory (system failure). This can result from an abnormal shutdown of the system or a programming error that cannot be recovered from using the state of main memory. In this case, the system is recovered using the state of the primary disk(s). This is known as "system recovery." The need for system recovery is detected by the database code and is performed automatically. Typically, system recovery should not seriously delay the execution of transactions.

The third failure condition is a primary disk error (media failure). Examples of this are a physically damaged hard drive or an abnormal shutdown that results in a partial disk write. In this case, the system is generally recovered using the state of a back up storage device such as a non-
primary disk. This is known as "media restore." Media restore is initiated by a user with the
rubix.restore.create.dbname authorization and can delay transaction processing.

6.2.2 Logging

In TR, logs are used to record operations that have been submitted to the database. These logs
are used to restore the database to a consistent state after one of the previously mentioned failure
conditions occur. TR has two types of logs:

- the rollback/recovery log
- the restore redo log.

The rollback/recovery log is used to recover from transaction and system failures. One such log is
maintained for each database and the files corresponding to this log are stored in the database
directory. The restore redo log is maintained in an administrator-defined location and is used to
recover from a media failure. The default location is also in the database directory. The state of
each log is placed on persistent storage at proper times in order to guarantee the database can be
placed back in a correct state following a failure.

6.2.3 Transaction Recovery

Transaction recovery is performed on-line automatically by the database system as transaction
failures occur. Rollback is executed by reading back through the rollback/recovery log and
removing the effects of all changes the transaction made to the database. As will be discussed in
the next section, sufficient disk space must be available for maintenance of the rollback/recovery
log.

6.2.4 System Recovery

System recovery is performed automatically on a per-database basis as the database is initially
opened. During the recovery process, no transactions may execute. The recovery process takes
the current state of the database on persistent storage along with the rollback/recovery log and
produces a correct database state reflecting the updates of all transactions that committed prior to
the system failure.

**WARNING**

System recovery requires that there be sufficient disk space to store the rollback/recovery
log. If this is not so, then recoverability cannot be guaranteed. Because each operation that
changes the state of the database is logged, the size of the log can be large. When a database
is closed normally, Trusted Rubix deletes the rollback/recovery log for that
database. Therefore, one way to control the size of the log is to periodically close and re-
open the database.

6.2.5 Media Restore (rxrestore)

Media restore is the responsibility of the user(s) with the rubix.restore.* authorizations. Prior to a
media failure, an image of the database is dumped to a non-primary disk using the rxdump
command. If a media failure occurs, the media is first checked for fatal defects and, if found,
replaced. The previously dumped database image is then restored and the contents of the restore
redo log are added.
6.3 Backup

It is important to consider accidental data loss and/or loss of data integrity, while maintaining any computer system. TR comes with a set of utilities to recover from catastrophic loss. A user holding the rubix.restore.backup.* authorization can fully backup databases at selected intervals. A user holding the rubix.restore.logs.set.* can also configure the logging of all database transactions between full system backups.

In TR, Operators are assigned the task of performing a full backup. Any facility using TR has to assess which databases are important and should be backed up. Anything that you can’t do without should be backed up.

It is also advisable to archive each backup to keep a historical record of the state of the database(s). The archive produced can be kept as a permanent record, and can be used at a later date should the need arise. The full backup schedule that is selected is determined by database activity. If the database in question is very dynamic, then backups should be more frequent than for an equivalent relatively static database.

<table>
<thead>
<tr>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>The full backup schedule that is selected should reflect database activity. Theoretically it is possible to safeguard the entire database from the time of its creation until its destruction with nothing but restore logging. Although possible, this strategy is discouraged. It would be impractical to restore a database using the logging information alone due to processing time constraints. If the database in question is very dynamic, then backups should be more frequent than for an equivalent relatively static database. Another factor in backup scheduling is the natural quiescence time of the system. Although, the archive system supplied by Trusted Rubix allows a backup to be performed at any time on a fully operational database, it is advisable to schedule backups during slow times. Such times as the midnight shift and weekends are prime candidates for full backups. Not only will the backup proceed more quickly, but users will be less likely to be inconvenienced by possible slowdowns caused by massive data movement. The last factor that the TR administrator should take into account is the availability of resources. Prime times are less likely to have resources such as a secondary disk drive and an operator with the proper authorization to conduct a full backup. In all cases, a regular backup schedule should be maintained. This will reduce the effort of restoring information and making sure information, is in fact, saved.</td>
</tr>
</tbody>
</table>

Each backup has an associated dump name, which may be an automatically generated value or specified as an option to the rxdump command. The automatically generated dump name has a format of RXDUMP-DBNAME-YR-MON-DAY-SEQNUM, where DBNAME is the name of the database being backed-up, YR-MON-DAY is the numerical year, month, and day of the backup, and SEQNUM is a sequence number used to distinguish multiple backups performed on the same day. The dump name is used to create a storage directory to contain the dump files and to refer to the dump when using the rxrestore command. The dump name storage directory must not exist prior to the rxdump invocation.

The backup is stored in one or more dump files named using a format of RXDUMP-SEQNUM.dmp, where SEQNUM is a sequence number. All backup files for a single backup are
stored within the DUMP_PATH/DUMP_NAME directory. Because multiple dump files may be used during the backup, there is no database size limitation for a backup to be performed. Data within the dump files may optionally be compressed. Use of compression may significantly reduce the size of a backup and is highly recommended. Data within the dump files is verified during rxrestore using a 32 bit cyclic redundancy check (CRC) code.

Internally, the rxdump command creates a database transaction and then reads through all objects in the database, writing them to the dump files. It therefore creates a snapshot of the database as of the created transaction. The transaction used to perform the backup is identified by the latest consistent moment (LCM), which is output as a date-time during the backup process. If the dump is used to restore the database, using the rxrestore command, the new database will be equal to the snapshot defined by the LCM. To apply updates that occurred after the lcm to the new database, restore redo logs must be applied during the restore process.

This command should be run at the SYS_HIGH MAC label to ensure that all objects are read. A warning message will be printed if the command is run at a label lower than SYS_HIGH. If a lower label is used, then only dominated objects will be included in the backup. Note that this may create errors during a restore that applies redo logs as updates to higher level objects will fail because the target objects will not exist in the restored database.

If a dump is archived, the storage directory (named by the dump name), all contained dump files, and any relevant restore redo log files should be included in the archive.

To better convey the TRR archive methodology, it will be described in the following components:

- Latest Consistent Moment (LCM) → Full Database Restore (rxrestore)
- Log Book → Manage Logs (rxlogs)
- Full Database Backup (rxdump)

**WARNING**

Physical secure countermeasures must be used to ensure backup media are safeguarded from destruction or theft. Backup media must be protected at the highest level of classification of the data on the media.

### 6.3.1 Latest Consistent Moment

All backup functionality is centered around the concept of the latest consistent moment (LCM). Any transactions which are later than the backup transaction are not reflected in the backup. For this reason, the database can be active during a full backup. TRR maintains the LCM, and displays it on the operator console when backups are initiated.

### 6.3.2 Log Book

For efficient and timely restorations it is important that an operator have immediate access to when and where backups and transaction logs were made. This may take the form of an on-line table or a backup logbook.
Each entry in the log book should be labeled minimally with the name of the database, date and time it was created and its latest consistent moment. This information is available from the output of the \textit{rxdump} command. This information is used during database restoration.

### 6.3.4 Full Database Backup

This section deals with the usage of \textit{rxdump} and the logging commands to save data disk. Most of the operations of \textit{rxdump} do not operate at a high level of data abstraction, because it is more efficient to deal directly with binary information from the lower level database functions. The command must be issued at the highest level at which data will be retrieved. Thus, the invoker must be \textsc{system high} to guarantee the backup of all data. A warning message will be printed if the command is run at a sensitivity label lower than \textsc{system high}.

\textit{rxdump} dumps exactly one database to dump location. All objects in the database will be backed up as of the LCM from the start of the program; only objects whose creation timestamp is less than or equal to the LCM will be written. For this reason a perfectly consistent backed-up dataset can be attained while the \textsc{trusted rubix} system is up and running. Only a user with the \texttt{rubix.restore.backup.dbname} authorization may perform this command.

→ **Steps:**

1. Determine the target file system for the dump and ensure that is has enough free space to store the dump.
2. Issue the \textit{rxdump} command as the Trusted RUBIX Operator.
3. Record the fact that this job has been performed in the official logbook. Remember to record time, by whom, and the latest consistent moment (LCM).
4. Clean up the old log files. Log files that consist entirely of records with timestamps prior to the LCM of the dump should be archived or deleted.
5. Archive (e.g., to DVD) any previous dumps as required by your operational requirements.

### 6.3.5 Restore Logging

\textsc{trusted rubix} full backups are useful for bulk writings, but after a few moments of on-line activity are out-of-date. Even if the database is slightly out of date, data safety cannot be assured. The solution is restore redo logging. Every SQL operation which modifies the database is logged through the \textsc{tr} restore redo logging system.

Logging is performed by the Log Module in the Common Server Subsystem. The log records are written to a TOS sequential file created in the log directory. Each file consists of a header and a set of log records. The name of the sequential file is:

\begin{verbatim}
It is suggested that the logbook not be on-line (or that frequent printouts or backups occur to that table). This precaution averts the danger of having a damaged online file \textit{and} a corrupt database to restore; they usually occur at the same time.
\end{verbatim}
log-YYYY-MM-DD-NNN

where YYYY is the year, MM is the month, DD is the day, and NNN is a sequential number starting at 001 and increments for each file created. The current log is closed and a new log file is opened when the maximum file size is reached.

→ Updating Logging Parameters

The user is allowed to change information concerning the logging of an existing database.

The rxlogs command gives authorized users the ability to list, delete, move, and set audit and restore log files for a specific database. The log files are created as operations that are performed on a database. Over time these files may need to be removed to free up disk space.

rxlogs - [a|r] [l|d FILENAME|ALL] [m|s NEWPATH] DATABASE

The user has the ability to change the maximum size of the log files. By setting a small size, the number of records in the file is small. If the number of records in the file is small, there is a chance that the timestamps of the records are in sequence. This makes handling old log files easier. If the timestamps of the records are in sequence, the file can be deleted or archived when the dump’s LCM is greater than the close time of the log file. If you have large log files, the range of timestamps is greater and the life of the log file increases. The value of SIZE must be at least 50K bytes.

The user can specify a shadow directory for logging. This causes the same log file to be written in both the log directory and the shadow directory. The benefit of using a shadow directory is that it provides redundant storage. The shadow directory should exist on a different disk than the database. This increases the likelihood of being able to recover the database in the event of a disk crash. If the disk with the database does crash, the integrity of the log files stored on the same disk as the database becomes questionable. Thus, in the event of a disk with the database crashing, the user can rebuild the database using the dump and the log files in the shadow directory.

→ Type of Logging

Logging information is saved on a per-database basis. In other words, every active database with logging enabled on the system has its own transaction logging output. The implication behind this assumption is that there are separate logging directories and files for each database.

6.4 Restore

This section deals with the restoration of data. This is a two-fold process in which the backup is restored and then, overlaid with the transaction logging entries. Only those with the rubix.restore.create.dbname are allowed to perform this operation.

WARNING

The database must remain off-line until the entire restoration process is completed. If the database is brought on-line while a restoration is underway, there is a substantial risk of corrupting the database and the loss of database integrity. The entire roll forward operation must be completed before any new transactions can be applied.
The roll forward is part of the restore process. By default, log files are stored in the database directory. If other log files are to be used, the directory where the log files reside is passed to the `rxrestore` command as a command line option.

After the database has been restored from the dump, the log directory is searched for log files. Each log file is applied to the database. When all the log files have been applied, the `rxrestore` command ends and the database is restored and ready to use.

Restoring data is a two-fold process in which the backup is restored and then, overlaid with the restore logging entries. Only those with the `rubix.restore.create.dbname` (or an authorization which dominates this permission, such as `rubix.restore.*`) are allowed to perform this operation.

`rxrestore` restores exactly one database from the specified file into the current directory. The database name must not already exist as defined in the current directory. Only users with the `rubix.restore.create.dbname` are allowed to use this command.

For more information on the `rxrestore` command refer to the Administrative Commands Reference Guide.

→ Steps:

1. Locate the dump to be used for the restoration and its name.
2. Issue the `rxrestore` command. You will need to specify the name of the new database (`-d` option) and the name of the dump to be used (`-n` option). If the dump is located in a non-standard path, the path must also be specified (`-p` option).
3. If restore redo log is to be applied during the restoration it must be specified to the `rxrestore` command. It may be specified that the default restore redo log location is to be used (`-r` option) or an explicit location may be specified (`-R` option).
4. Record the fact that the restoration has been performed in the official Trusted Rubix logbook. Remember to record time, by whom, and to what directory.

### WARNING

Do not delete any information (especially removing the database) until the new restoration is complete and verified.

No other application or user should be using the given database during the restoration. As enforcement, each of the relations in the database will be locked exclusively during its restore operation.
6.5 Row Reclassification

The ability to change the sensitivity label of stored rows in a table (using the SQL UPDATE command to update the rowlabel field) is a critical component.

In an MLS environment, it is necessary for security officers, administrators and other site-specified privileged users to perform administrative functions (like performing a full database export and import, or relabeling of row information). To enable bypassing of the MAC policy of the TOS (without breaking the overall security policy), it is necessary to provide facilities for privileged user's to read up (read higher sensitivity label data), write up (write to higher sensitivity label data), write down (write to lower sensitivity label data) and write across (write to an incomparable sensitivity label) within the database. This process of reclassification of data is described by three operations:

→ Upgrade - raise the sensitivity label of the data
→ Downgrade - lower the sensitivity label of the data
→ Across - change the sensitivity label of the row to an incomparable label

Trusted Rubix associates each database object on the system with a sensitivity label. This sensitivity label is stored in the hidden ‘ROWLABEL’ column associated with each row. When the object is created, the sensitivity label of the object is set to the sensitivity label of the subject; i.e., the session sensitivity label of the user who created the object.

To enable the manipulation of sensitivity labels as part of the MAC policy, it is necessary to have one of the rubix.mac.recl.* authorizations.

Re-classification of rows is accomplished by a TR Security Administrator issuing an update on the rowlabel field using the SQL UPDATE command.

Row re-classification in TR is governed by the polyinstantiation method used when the table was created. In the POLYNONE case, any row can be re-classified, because the row with the specified primary key is unique. In the POLYHIGH case, the re-classification is rejected if the table already contains a row with the same primary key at the new level. In the POLYLOW case, the re-classification is rejected if the table already contains a row with the same primary key at a level equal to or dominated by the new level.

6.6 Miscellaneous

6.6.1 Drop, list, or rename a database

The rxdb command is used to list, drop, or rename databases. rxdb may be used to drop a database even when the database is corrupted and will not open.

To list a database the database sensitivity label must be dominated by the user's TOS session sensitivity label and the user must have the rubix.admin.db.ls.dbname authorization. To move (rename) a database the database sensitivity label must equal the user's TOS session sensitivity label and the user must have the rubix.admin.db.mv.dbname authorization. To drop a database the user must have the rubix.admin.db.rm.dbname authorization.
For more information on the \texttt{rxdb} command, see the \textit{Administrative Commands Reference Guide}.

\subsection*{6.6.2 Database Storage Locations}

A database may store significant amounts of data on the server platform. Therefore, care should be used in choosing where storage files are located. Distributing the data across multiple disks may also improve performance. This is especially true for the recovery log and volume files in a high update environment.

Administrator may choose where audit logs, restore logs, recovery logs, volume files (regular table data), and LOB files (large object data types) are stored. There are two ways in which a storage directly may be specified. The first is by specifying a configuration option in the \texttt{rxconfig} file prior to creating a database. For information on this please the section titled \textit{Trusted RUBIX Configuration} in this document.

Once a database has been created the storage directories may be moved using with the \texttt{rxlogs} command (audit, restore, and recovery logs) or the \texttt{rxdb} command (volume and LOB storage directories). Please see the \textit{Trusted RUBIX Administrative Commands Reference Guide} for more information.
CHAPTER 7   AUDIT ADMINISTRATION

This chapter discusses the administration of the Trusted Rubix audit subsystem. The Trusted Rubix audit subsystem provides a means to assure that all users and administrators are accountable for their actions within TR, in accordance with the Common Criteria for EAL4. We begin with a description of events in the TR audit trail and the fields saved for each audit event.

The events to be audited can be broken into two parts. The first part is handling interaction with the underlying Trusted Computing Base (TCB). The auditing of these events is done by the TCB. Whenever the mandatory or discretionary security policy is invoked, the event needs to be logged in the audit trail.

Second, is the capability for an authorized person to access and evaluate the accountability information by a secure means. The system must audit any action that can potentially cause access to, generation of, or cause the release of classified or sensitive information. The audit data is selectively acquired based on the auditing needs of the installation and/or application. The audit data must be captured with enough granularity to trace the events to a specific individual.

7.1 Audited Events

In order to meet the Common Criteria EAL4 requirements, TR needs to audit the introduction and deletion of objects. The Audit Administrator can selectively audit the actions of any one or more users based on individual identity and/or object sensitivity label. The following types of event are recorded by TR:

- introduction of subjects to TR;
- introduction of objects into a user's address space;
- deletion of objects;
- actions taken by computer operators and other administrators;
- operations (updates, retrievals, and inserts) initiated by untrusted users;
- discretionary and mandatory access control policy decisions made by TR;
- creation and modification of ACLs;
- use of commands in an administrative role;
- other security relevant events.

There are certain auditable events that may occur automatically when the user executes a command or SQL statement that may not seem directly related to the auditable events. Those events are:

- open database,  commit transaction,  cursor open and
- create transaction,  rollback transaction,  cursor close.

Any time a user issues any SQL operation on a database or executes the import and export commands the database is opened. This may result in the database open being audited. The audit data for this operation is:

- event name (sql_db_open),  session sensitivity  process ID,
When a user performs any operation on the database which requires transactions, a transaction is automatically created. Furthermore, depending on the configuration of the RXISQL client any open transactions will be automatically rolled back or committed when the RXISQL program terminates. The audit data for these events is:

- Event name (sql_trans_create, sql_trans_commit, sql_trans_rollback)
- User ID
- Group ID
- Current database name
- Operation status
- Timestamp
- Transaction ID
- Target database label
- Target database name

When a user issues a SELECT statement on the database a cursor is created which represents the structure of the SELECT statement. This cursor may represent arbitrary queries on multiple tables and views. The cursor defines both the physical objects selected from and the particular columns actually viewed. Trusted RUBIX only audits those columns which the user actually sees as a result of his query. When the SELECT statement is issued a cursor is created and the cursor is removed upon session termination. These actions may result in a cursor open or cursor close audit event. The audit data for cursor open is:

- Event name (sql_cursor_open)
- User ID
- Group ID
- Database name
- Session sensitivity label
- Operation status
- Timestamp
- Transaction ID
- Process ID
- Session ID
- Cursor label
- Current catalog name
- Current schema name
- Cursor name
- Cursor definition

The audit data for cursor close is:

- Event name (sql_cursor_close)
- User ID
- Group ID
- Database name
- Session sensitivity label
- Operation status
- Timestamp
- Transaction ID
- Process ID
- Session ID
- Cursor label
- Cursor name

### 7.2 Trusted RUBIX Audit Events

TR audits all security relevant actions relative to the objects which it manages. It also audits other internal actions which may have security relevant side effects. The security-relevant actions audited by TR are listed below.

They are grouped by the type of objects manipulated:
### Database Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Database</td>
<td>sql_db_create</td>
<td>an attempt was made to create a database (e.g., via the CREATE DATABASE command)</td>
</tr>
<tr>
<td>Drop Database</td>
<td>sql_db_drop</td>
<td>an attempt was made to destroy a database (e.g., via the rxdb command)</td>
</tr>
<tr>
<td>Open Database</td>
<td>sql_db_open</td>
<td>an attempt was made to open a database. The database is opened prior to any SQL command being executed on that database. The database is also opened when rximport and rxexport are issued.</td>
</tr>
</tbody>
</table>

### Catalog Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Catalog</td>
<td>sql_catalog_create</td>
<td>an attempt was made to create a catalog (e.g., via the CREATE CATALOG command)</td>
</tr>
<tr>
<td>Drop Catalog</td>
<td>sql_catalog_drop</td>
<td>an attempt was made to destroy a catalog (e.g., via the DROP CATALOG command)</td>
</tr>
</tbody>
</table>

### Schema Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Schema</td>
<td>sql_schema_create</td>
<td>an attempt was made to create a schema (e.g., via the CREATE SCHEMA command)</td>
</tr>
<tr>
<td>Drop Schema</td>
<td>sql_schema_drop</td>
<td>an attempt was made to destroy a schema (e.g., via the DROP SCHEMA command)</td>
</tr>
</tbody>
</table>

### Defined View Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create View</td>
<td>sql_view_create</td>
<td>an attempt was made to create a defined view (e.g., via the CREATE VIEW command)</td>
</tr>
<tr>
<td>Drop View</td>
<td>sql_view_drop</td>
<td>an attempt was made to remove a defined view (e.g., via the DROP VIEW command)</td>
</tr>
</tbody>
</table>

### Relation Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Relation</td>
<td>sql_rel_create</td>
<td>an attempt was made to create a relation within a database (e.g., via the CREATE TABLE command)</td>
</tr>
<tr>
<td>Drop Relation</td>
<td>sql_rel_drop</td>
<td>an attempt was made to remove a relation from within a database (e.g., via the DROP TABLE command)</td>
</tr>
</tbody>
</table>
## Relation Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alter Relation</td>
<td>sql_rel_alter</td>
<td>an attempt was made to change the structure of a relation (e.g., via the ALTER TABLE command)</td>
</tr>
</tbody>
</table>

## Index Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Index</td>
<td>sql_idx_create</td>
<td>create an auxiliary index for a relation (e.g., via the CREATE INDEX command)</td>
</tr>
<tr>
<td>Drop Index</td>
<td>sql_idx_drop</td>
<td>destroy an auxiliary index for a relation (e.g., via the DROP INDEX command)</td>
</tr>
</tbody>
</table>

## Transaction Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Transaction</td>
<td>sql_trans_create</td>
<td>an attempt was made to start a new transaction (e.g., via the SET TRANSACTION command)</td>
</tr>
<tr>
<td>Commit Transaction</td>
<td>sql_trans_commit</td>
<td>an attempt was made to terminate a transaction (e.g., via the COMMIT command)</td>
</tr>
<tr>
<td>Rollback Transaction</td>
<td>sql_trans_rollback</td>
<td>an attempt was made to terminate a transaction (e.g., via the ROLLBACK command)</td>
</tr>
<tr>
<td>Savepoint Declare</td>
<td>sql_svpt_declare</td>
<td>an attempt was made to establish a named savepoint within a transaction (e.g., via the SAVEPOINT command)</td>
</tr>
<tr>
<td>Savepoint Release</td>
<td>sql_svpt_release</td>
<td>an attempt was made to release a previously declared savepoint (e.g., via the RELEASE SAVEPOINT command)</td>
</tr>
<tr>
<td>Savepoint Rollback</td>
<td>sql_svpt_rollback</td>
<td>an attempt was made to undo all updates performed since the specified savepoint, while at the same time preserving updates performed prior to that point (e.g., via the ROLLBACK TO SAVEPOINT command)</td>
</tr>
</tbody>
</table>

## Row Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Row</td>
<td>sql_row_create</td>
<td>an attempt was made to insert a new row into a relation (e.g., via the INSERT command)</td>
</tr>
<tr>
<td>Delete Row</td>
<td>sql_row_delete</td>
<td>an attempt has been made to delete a row from the current representation of a relation (e.g., via the DELETE command)</td>
</tr>
<tr>
<td>Update Row</td>
<td>sql_row_update</td>
<td>an attempt has been made to modify one or more elements of a row within a relation (e.g., via the UPDATE command)</td>
</tr>
</tbody>
</table>
## Miscellaneous Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Modify</td>
<td>sql_acl_modify</td>
<td>an attempt has been made to modify a privilege to a subject (e.g., via the GRANT command)</td>
</tr>
<tr>
<td>Fetch</td>
<td>sql_fetch</td>
<td>an attempt has been made to fetch a row from a table (e.g., via the SELECT command)</td>
</tr>
<tr>
<td>Cursor Open</td>
<td>sql_cursor_open</td>
<td>an attempt has been made to open a cursor (e.g., via the SELECT command)</td>
</tr>
<tr>
<td>Cursor Close</td>
<td>sql_cursor_close</td>
<td>an attempt has been made to close a cursor (e.g., via the SELECT command)</td>
</tr>
</tbody>
</table>

## Administrative Audit Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Modify</td>
<td>sql_audit_modify</td>
<td>an attempt has been made to modify the auditable events, turn auditing on, or turn auditing off using rxauditset.</td>
</tr>
<tr>
<td>Audit Review</td>
<td>sql_audit_review</td>
<td>an attempt has been made to review the audit trail using rxauditrpt.</td>
</tr>
<tr>
<td>Import</td>
<td>sql_import</td>
<td>an attempt has been made to import data using rximport.</td>
</tr>
<tr>
<td>Export</td>
<td>sql_export</td>
<td>an attempt has been made to export data using rxexport.</td>
</tr>
<tr>
<td>Dump</td>
<td>sql_dump</td>
<td>an attempt has been made to backup a database using rxdump.</td>
</tr>
<tr>
<td>Restore</td>
<td>sql_restore</td>
<td>an attempt has been made to restore a database using rxrestore.</td>
</tr>
<tr>
<td>Alter Session Set Label</td>
<td>sql_lbl_alter</td>
<td>an attempt was made to alter the session sensitivity label (e.g., via the ALTER SESSION SET LABEL command)</td>
</tr>
<tr>
<td>Reclassify Row</td>
<td>sql_recl_row</td>
<td>an attempt was made to reclassify a row.</td>
</tr>
<tr>
<td>Manage Log Files</td>
<td>sql_logs</td>
<td>an attempt was made to manage audit or restore log files using the rxlogs command</td>
</tr>
<tr>
<td>Drop, List, Move (Rename) Database</td>
<td>sql_db</td>
<td>an attempt was made to drop, list, or move (rename) a database using the rxdb command.</td>
</tr>
<tr>
<td>Manipulate SPM Policies</td>
<td>sql_polman</td>
<td>an attempt was made to manipulate the SPM policies using the rxpolman command.</td>
</tr>
<tr>
<td>SPM custom audit event</td>
<td>sql_spm_event</td>
<td>A custom SPM audit event has occurred.</td>
</tr>
</tbody>
</table>

### 7.3 AUDIT DATA

The audit data consists of two types of data. The first type is audit information common to all audit records. The second type is event specific data and varies according to the event being audited. The following tables list the information that is audited. It should be noted that the order of information, which appears in the tables, is the same order that data appears in the audit record.
Information enclosed in brackets is optional, depending upon the specific operation being audited.

<table>
<thead>
<tr>
<th>Common Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event name</strong></td>
</tr>
<tr>
<td><strong>User ID</strong></td>
</tr>
<tr>
<td><strong>Group ID</strong></td>
</tr>
<tr>
<td><strong>Database name</strong></td>
</tr>
<tr>
<td><strong>Session sensitivity label</strong></td>
</tr>
<tr>
<td><strong>Operation status</strong></td>
</tr>
<tr>
<td><strong>Timestamp</strong></td>
</tr>
<tr>
<td><strong>Transaction ID</strong></td>
</tr>
<tr>
<td><strong>Process ID</strong></td>
</tr>
<tr>
<td><strong>Session ID</strong></td>
</tr>
</tbody>
</table>

The common information saved for all audit records is:

<table>
<thead>
<tr>
<th>Database Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Database</td>
</tr>
<tr>
<td>Drop Database</td>
</tr>
<tr>
<td>Open Database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Catalog Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Catalog</td>
</tr>
<tr>
<td>Drop Catalog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schema Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Schema</td>
</tr>
<tr>
<td>Drop Schema</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Defined View Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create View</td>
</tr>
<tr>
<td>Drop View</td>
</tr>
<tr>
<td>Relation Events Audit Information</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Relation</td>
</tr>
<tr>
<td>Drop Relation</td>
</tr>
<tr>
<td>Alter Relation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Index</td>
</tr>
<tr>
<td>Drop Index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Transaction</td>
</tr>
<tr>
<td>Commit Transaction</td>
</tr>
<tr>
<td>Rollback Transaction</td>
</tr>
<tr>
<td>Savepoint Declare</td>
</tr>
<tr>
<td>Savepoint Release</td>
</tr>
<tr>
<td>Savepoint Rollback</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>Create Row</td>
</tr>
<tr>
<td>Delete Row</td>
</tr>
<tr>
<td>Update Row</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous Events Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event</strong></td>
</tr>
<tr>
<td>ACL Modify</td>
</tr>
<tr>
<td>Fetch</td>
</tr>
<tr>
<td>Cursor Open</td>
</tr>
<tr>
<td>Cursor Close</td>
</tr>
</tbody>
</table>
### Administrative Events Audit Information

<table>
<thead>
<tr>
<th>Event</th>
<th>Audit Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Modify</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Audit Review</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Import (per row)</td>
<td>relation label, catalog name, schema name, relation name, unix import file name, row data</td>
</tr>
<tr>
<td>Export (per row)</td>
<td>relation label, catalog name, schema name, relation name, unix export file name, row data</td>
</tr>
<tr>
<td>Dump</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Restore</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Alter Session Set label</td>
<td>new session label</td>
</tr>
<tr>
<td>Reclassify Row</td>
<td>old row label, new row label, catalog name, schema name, relation name, unix input file, row data</td>
</tr>
<tr>
<td>Manage Log Files</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Drop, List, or Move (Rename) a Database</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Manipulate SPM policies</td>
<td>database label, command line arguments</td>
</tr>
<tr>
<td>Custom SPM event</td>
<td>Dependent on the configuration of the security policy obligation creating the custom event</td>
</tr>
</tbody>
</table>

### 7.3.1 ACL Information

The ACL information has five fields:

- User ID of grantor
- bitmap of privileges held by the grantee
- bitmap of privilege that grantee may give to others
- User ID of grantee
- Group ID of grantee

A bitmap of ACL privileges consists of an “OR”ing of some number of the following values:

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>0X0001</td>
</tr>
<tr>
<td>references</td>
<td>0X0002</td>
</tr>
<tr>
<td>select</td>
<td>0X0004</td>
</tr>
<tr>
<td>update</td>
<td>0X0008</td>
</tr>
<tr>
<td>crview</td>
<td>0X0010</td>
</tr>
<tr>
<td>delete</td>
<td>0X0020</td>
</tr>
<tr>
<td>null</td>
<td>0X0100</td>
</tr>
<tr>
<td>refview</td>
<td>0X0200</td>
</tr>
<tr>
<td>grantnull</td>
<td>0X0400</td>
</tr>
<tr>
<td>admin</td>
<td>0X8000</td>
</tr>
</tbody>
</table>

**7.4 Audit Administration (rxauditset and rxauditrpt)**

The **Trusted Rubix** audit administrator is responsible for managing the **TR** audit
subsystem in a manner that ensures that all TR users are held accountable for their actions in accordance with the assurance requirements of the Common Criteria EAL 4 Level and the security policy of their organization. Auditing of database operations is performed on TR server machine. The TR auditing subsystem provides the means by which the audit administrator may accomplish this goal. It consists of four components:

1. audit event detection and recording functions
2. a utility for managing which auditable events are to be recorded
3. a utility for examining the recorded audit events
4. procedures for managing audit data

The TRUSTED RUBIX auditing subsystem provides tools which allow an Audit Administrator (or auditor) to specify which audit events are to be recorded and to analyze the recorded data.

The utility used to specify the (pre-selection) criteria, which determine the auditable events to be recorded, is the rxauditset command. This utility allows the auditor to specify which events are to be recorded on a system-wide basis, as well as those to be recorded for specific users, for TR objects residing at a specific sensitivity label and for TR objects residing within a range of sensitivity labels. The utility used to specify the (post-selection) criteria, which determine the recorded events to be displayed, is the rxauditrpt command. It allows an auditor to specify which recorded events are to be displayed based on a variety of criteria.

TR attempts to record an event when it detects that a security relevant action has taken place which warrants auditing. When TR calls the audit event recording function, it determines if the event is auditable based on whether it meets the pre-selection criteria established by the audit administrator. This function then attempts to record the relevant information about the subject, object(s), and other event specific information into the TR audit trail using the TR audit recording functions. If any operation fails to write required audit data to the auditing trail the following series of events happen:

→ All active transactions are rolled back;
→ The server process closes all open objects and exits;
→ The client is given an audit record storage error code.

TR provides the audit administrator with the ability to specify which events are to be selected for recording by the audit recording function through the definition of pre-selection criteria. When a security relevant event occurs which meets the pre-selection criteria, it is recorded into the TR audit trail. It is then possible for the auditor to obtain a subset of previously recorded audit data through the use of post-selection criteria.

7.4.1 Pre-Selection Criteria

The audit subsystem provides an audit administrator with the capability to limit the amount of audit data collected. This is accomplished by allowing the auditor to specify criteria which allow the TR audit subsystem to select which audit events are to be recorded in the audit trail. The criteria (known as pre-selection criteria) available to an audit administrator to specify which events are to be recorded include the following:
→ events to be recorded system-wide at all times
→ events to be recorded based on the event types
→ events to be recorded based on the identity of the user/group performing the action
→ events to be recorded based on the sensitivity label of the subject
→ events to be recorded based on the sensitivity label of the object(s) involved
→ events to be recorded based upon the database being operated upon.
→ turn on or off the auditing system without effecting the other audit criteria

As mentioned earlier, the utility for managing the pre-selection criteria is the `rxauditset` command. This command allows an auditor to specify the events which are to be recorded on a system-wide basis, for a specific user/group or set of users/groups, and for a specific object or subject sensitivity label and/or range of sensitivity labels.

The `rxauditset` command allows a user with the appropriate privileges to specify or display the criteria which determine whether an auditable event is eligible to be recorded by Trusted Rubix in the audit trail. Any updates made to the audit event list will not affect existing server processes. All new invocations of the database server will use the updated audit event list.

The following examples illustrate the implementation of common audit tasks.

Audit criteria must often be set to monitor access to sensitive information. The following two commands will initiate the monitoring of accesses of “TS” data:

```
rxauditset -a sql_fetch
rxauditset -m -L TS sql_fetch
```

The first command initiates (adds) the global monitoring of all data accesses. The list of audit events is then modified by the second command to monitor only “TS” sensitivity label objects on which a fetch is performed. By using these two commands the Audit Administrator can identify all users who attempted to access information with the “TS” sensitivity label.

Another common task is to audit the creation of rows of a table. The following two commands would add an audit event for the creation any row of information in a table:

```
rxauditset -a sql_db_create
rxauditset -m -R C-TS sql_db_create
```

As in the previous example, the first command initiates (adds) the global monitoring of all database creation events. The second command modifies the audit event to only monitor the creation of databases with the “C” through “TS” sensitivity label. The use of the “-R” (range) option is a short hand for the following three commands:

```
rxauditset -m -L TS sql_db_create
rxauditset -m -L +S sql_db_create
rxauditset -m -L +C sql_db_create
```

Other common events to audit are database creation, dropping a database, creating a table (relation), and dropping a table (relation). All of these audit events can be defined globally with the following command:
rxauditset -a sql_db_create,sql_db_drop,sql_rel_create,sql_rel_drop

Sometimes it is desirable to audit a particular user. The following two commands would add a monitoring event for changes to the DAC privilege for the specific users “mary” and “bob”:

```
rxauditset -a sql_acl_modify
rxauditset -m -u mary,bob sql_acl_modify
```

As before, the first command creates an audit event for all ACL modifications, and the second command modifies the audit event list to only audit the specific users “mary” and “bob”.

For more information about the use of the `rxauditset` command, see the description of the `rxauditset` command in the Administrative Commands Reference Guide.

### 7.4.2 Post-Selection Criteria

**Trusted Rubix** provides an audit data reduction utility which allows the auditor to limit the scope of data extracted from the audit trail. The criteria (known as post-selection criteria) available to an audit administrator to specify which events are to be extracted from the audit trail include the following:

- event type
- event status (success/failure)
- identity of the user performing the action
- sensitivity label of the subject performing the action
- sensitivity label of the object(s) involved
- time of first record to extract
- time of last record to extract

The utility which provides this capability, the `rxauditrpt` command, gives the auditor the ability to specify the criteria which identify the set of records to be selected for extraction.

The Audit Administrator can generate a report of all row fetches that have been audited with the following command:

```
rxauditrpt -e sql_fetch
```

These commands can be precise:

```
rxauditrpt -u root -l U -t 2006-01-31-12-08-01:2007-01-31-13-08-01 -e sql_acl_modify
```

The above command reports on DAC privileges granted or revoked by “root” during the specific interval of January 31, 2006 at 12:08:01 to January 31, 2007 at 13:08:01, and only users with a “U” sensitivity label.

The `rxauditrpt` utility can display audit information in an orderly manner if the user is a trained Audit Administrator. Sorting is not an explicit option of `rxauditrpt`. Sorting, however, can be done by invoking `rxauditrpt` multiple times with a combination of options. Sorting in the order of object sensitivity label can be done by invoking rxauditrpt with the ”-l” option and specifying Top Secret, for example:
This can be followed by invoking `rxauditrpt` with the "-l" option for Secret and Confidential as shown below:

```
rxauditrpt -l S
rxauditrpt -l C
```

These three commands, executed in succession, would list all audited actions at the "TS", "S" and "C" sensitivity label. By generating a report for each label in the database, a listing of the audit log contents for each sensitivity label can be produced.

This same method works for sorting the audit report by user labels ("-L" option) and different specified events ("-e" option) to display audit information in the order of the labels and events in each database.

For more information about the use of the `rxauditrpt` command, see the description of the `rxauditrpt` command in the **Administrative Commands Reference Guide**.

### 7.4.3 Audit Log Familiarization

Audit log information is specific to each event type recorded. The common audit information was previously shown, along with separate tables which give the event specific information.

Audit Administrators should consider auditing the creation of databases. A user who is collecting information will often do so in a private database. Therefore, the creation of a database is often a tip-off that a user warrants additional attention. The audit report for the creation of a database is given below:

```
rxauditrpt -e sql_db_create
sql_db_create
UID: cjfoo (2046)
GID: user (500)
Database: master
Session label: U
Operation Status: Success
Transaction ID: UNDEFINED
Process ID: 611
Session ID: 9908
Object label: U
Database: d
```

The first line gives the command for generating a report of all databases created. The following lines show the report produced by the command, given the events recorded by previous `rxauditset` commands. This report shows that database “d” was created in database `master`, and has a classification of “U.” Note that the `master` database is a special database automatically created and used by Trusteed Rubix for special operations such as creating and dropping databases.
Another common auditing task is to list all users who have accessed a particular object. This type of report is useful when a piece of information is mislabeled and it is necessary to determine the scope of the breach. This type of report can be generated as follows:

```
rxauditrpt -L TS -d d sql_fetch
sql_fetch
  UID: Mary (2041)
  GID: user (500)
  Database: d
  Session label: TS
  Operation status: Success
  Transaction ID: 1044051231.297888
  Process ID: 4746
  Session ID: 9908
  Object label: TS
  Catalog: default_catalog
  Schema: default_schema
  Cursor definition: DECLARE FIXED_NAME CURSOR FOR select * from t FOR READ ONLY
  Row data: 23

sql_fetch
  UID: Hal (2043)
  GID: user (500)
  Database: d
  Session label: TS
  Operation status: Success
  Transaction ID: 1044051973.318001
  Process ID: 12115
  Session ID: 19704
  Object label: TS
  Catalog: default_catalog
  Schema: default_schema
  Cursor definition: DECLARE FIXED_NAME CURSOR FOR select * from t FOR READ ONLY
  Row data: 23
```

As before, the command that generates the report is given in the first line. (The report assumes that the audit information was recorded, using an `rxauditset` command). This report shows that in database “d”, two users, Mary and Hal, accessed “TS” labeled information in table “t” in two separate transactions. The cursor definition shows that the entire contents of the table were accessed. Thus if there has been a breach of security by improperly releasing information at the “TS” sensitivity label in table “t”, it is limited to these two users.

Commonly, selected contents of a table will disappear. This type of problem can be isolated by reporting deletes done on the object table.
rxauditrpt -t 2003-01-31-15-00:2003-01-31-15-30 -d -t employee
sql_row_delete

sql_row_delete
    UID: Joe (2046)
    GID: user (500)
    Database: d
    Session label: U
    Operation status: Success
    Transaction ID: 1044044832.29340
    Process ID: 24591
    Session ID: 9908
    Object label: U
    Catalog: default_catalog
    Schema: default_schema
    Table: employee
    Row data: Frank

sql_row_delete
    UID: Joe (2046)
    GID: user (500)
    Database: d
    Session label: U
    Operation status: Success
    Transaction ID: 1044044832.29340
    Process ID: 24591
    Session ID: 9908
    Object label: U
    Catalog: default_catalog
    Schema: default_schema
    Table: employee
    Row data: Hal

The first line gives the command that lists all delete row operations for database “d” and table “employee”. One delete would be shown for every row deleted. In this case, user “Joe” deleted users “Frank” and “Hal”. It should be noted that user “Joe” can be prevented from performing deletes through manipulation of his DAC privileges. Assuming that the Database Administrator agrees, the following would report the removal of user “Joe” delete privilege on the “employee” table.

rxauditrpt -u Joe sql_acl_modify
sql_acl_modify
    UID: Joe (2046)
The command to produce a report of modifications to Joe’s ACL is shown in the first line. The Operation field shows that a revoke was performed. The Bitmap field value of “0x20” can be verified in the ACL list table to be the delete privilege that was revoked.

7.4.4 Maintenance of Audit Data

The database specific audit files are destroyed when that database is destroyed. Audit files may be garbage collected (selectively deleted) by the audit administrator by using the Trusted Rubix administrative command `rxlogs`, which allows the Audit Administrator to:

→ Delete audit files for a specific database that were created prior to a given date.
→ List the audit files and their sizes.
→ Set audit log files.

Upon detection of audit storage failure, the TR server process automatically terminates itself as the prevention of audit data loss.

WARNING

Since the TR auditing subsystem places audit events into the a TR audit file that is separate from the TOS audit file, it relies on an authorized TR user to properly manage and protect the audit data once it has been recorded.

See also the `rxlogs` command discussion in the Administrative Commands Reference Guide.
CHAPTER 8  DELIVERY AND INSTALLATION

This document outlines the steps taken to ensure that security is maintained when delivering **Trusted Rubix** to a customer site. It also describes the steps necessary for secure installation, generation, and start-up of **TR** at the customer site.

8.1 Delivery Strategies

As long as the **TR** software is within the walls of the controlled ITI facilities, it is assumed to be secure and the intended software for the customer. When the **TR** software has been installed successfully onto the customers target platform, it is also assumed to be the intended software. Issues arise during the transit between the two locations.

After the **TR** software leaves ITI, it must arrive intact at the customer site and be delivered internally to the intended recipient. ITI has no control over the customer procedures, so steps must be taken to verify that the software that leaves ITI arrives at the target host safely. In particular, ITI must be alert for man-in-the-middle and stolen identity/spoofing attacks.

8.1.1 Attack Strategies

ITI addresses two types of attack strategies:

- **man-in-the-middle attacks**
- **stolen-identity/spoofing attacks**

The man-in-the-middle attack occurs when software is purposely compromised during delivery such that a perpetrator can take advantage of the system. A compromise of this sort will most likely result in the software being altered in such a way as to provide a back door to the intended customer system, ITI software, or possibly cause damage to the intended site. None of these outcomes are acceptable.

The stolen identity/spoofing attack occurs when a perpetrator assumes the role of a trusted source and fools the customer into installing software not developed at ITI. The customer receives a package or software update that looks like it comes from ITI, but in fact is software developed with the intention of superseding the **TR** product. Again, this software will most likely be used to corrupt the customer site in some non-benign way.

This attack is arguably the most used method within the current Internet hacking environment and is most often associated with social engineering. Pretending to be someone else is very easy and may be associated with a high level of exposure for the victim.

8.1.2 Protection Technologies

The two candidate protection technologies are encryption and one-way hashing mechanisms (i.e., MD5). Both technologies make use of similar approaches, but are differentiated by ideological differences. Encryption is used to protect data during delivery and hashing is used to verify delivery. Since ITI’s goal is to ensure that delivery is verified, ITI has deployed a one-way hashing procedure.
Encryption is the process of scrambling data so as to prevent unauthorized access. Modern encryption techniques quite often employ private/public key schemes so that the secret used to encrypt the data is not revealed to the recipient. The technique of encryption is not that useful to the ITI delivery procedures because ITI is interested in only making sure that the software being installed is verified as the product that shipped from ITI, not in protecting a third party from reviewing the released deliverables.

Hashing produces a message digest which is a unique value that is algorithmically produced from the contents of a data set. The mechanism used to produce the message digest is well known and easily reproduced. The message digest produced by the MD5 hashing mechanism has the property that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. It is required that the customer request this value from ITI using their contact information. Disclosing the message digest does not compromise the verification of the secure delivery of Trusted Rubix software. Thus, disclosure of the message digest by ITI, or any customer who is in possession of it, does not pose a security risk.

8.1.3 Delivery

The first step in getting TR software safely and verifiably from ITI to the customer is to ensure that the product arrives at the customer’s site. ITI has taken the position that delivery via physical means is safer than a digital format. Time is not as critical an element as security. Thus, a conservative package delivery firm is utilized (e.g., FEDEX). Firms such as FEDEX are used daily to deliver large amounts of critical and sensitive data around the world. When TR arrives at the customer’s site, it must be signed for and accountability is thus preserved.

8.1.4 Intra-office delivery

Once TR arrives at the customer’s site, it must be delivered to the intended recipient. The delivery and receipt of packages are completely out of the control of ITI, so steps must be taken to verify the final delivery of TR through the use of hashing.

8.1.5 Verified Installation

Before installation onto the target host, the customer’s installer verifies possession of the correct software by contacting ITI by telephone or fax and requesting a message digest for the delivery. If ITI has scheduled a delivery of TR, a value is supplied to the customer. The customer runs the one-way hashing program at their site and compares it against the ITI supplied values.

If ITI has not scheduled a delivery or if the message digest values do not match then the product is considered invalid and must not be installed. An error in the media will be detected in this fashion as well as a malicious attempt to tamper with the TR software.

8.1.6 Message Digest Delivery

ITI can deliver the message digest to any interested party. When a customer calls for a value, it
will be delivered via any method deemed most useful at the time. The methods can be one of FAX or transcribed over the telephone. Because of the insecurities in standard email, standard email is not a valid alternative.

8.1.7 Customer Request

The customer needs to request a message digest once Trusted Rubix has arrived, but before attempting to install it. A faxed or a telephone transcribed MD5 message digest request is made by the customer.

8.1.8 Message Digest Verification

Insert the CD into the CDROM device and login in as any user who has privileges to read the device. Calculate the message digest for the candidate distribution for comparison against the expected ITI values by issuing the following command:

```bash
$ rxmd5 /dev/cdrom
```

If the output of the preceding command differs in any way from the expected ITI values, the CD media must not be installed.

8.2 Installation

For detailed instructions on how to install Trusted RUBIX please see the Trusted Rubix Installation and Quick Start Guide specific to your TOS.
CHAPTER 9  TRUSTED RUBIX CONFIGURATION

The Trusted Rubix Server configurable behavior is controlled through the rxconfig text file in the RUBIX_HOME/etc directory. Each configuration option has a defined minimum, maximum, and default value. If a configuration option has not been specified then its default value is used. If a value is specified that is outside the range of the minimum and maximum then it is ignored and the default value is used. If a value is specified more than once then the last occurrence is used.

Each configuration option is specified by a single line in the RUBIX_HOME/etc/rxconfig text file as “option_string = option_value” where option_string identifies the option being set. It may end with “.DBNAME” to specify a configuration option for a specific database. If an option string does not end with “.DBNAME”, then it is considered a global configuration option value and would apply to any database which does not have a database specific option value set. Database specific option values take precedence over global option values for the named database.

There are three types of configuration options: Boolean, integer, and string.

Boolean types represent options that may only be in one of two states, true or false. An example is using secondary authentication. Valid text values for Boolean types that are to be set to true are: “ON”, “TRUE”, and “1”. Valid text values for Boolean types that are to be set to false are: “OFF”, “FALSE”, and “0”. All alphabetic values are case insensitive. Integer types represent options that are numerical in nature, such as a buffer size. Integer text values may be specified in decimal or hexadecimal and must fall within the defined minimum and maximum range. A table defining each option and its minimum, maximum, and default values follows:
<table>
<thead>
<tr>
<th>String Value</th>
<th>Type</th>
<th>Default</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipc.lcktimeout</td>
<td>INT</td>
<td>60</td>
<td>0</td>
<td>360</td>
</tr>
<tr>
<td>ipc.ismshmem</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>buffer.numbuf</td>
<td>INT</td>
<td>2500</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>buffer.prstat</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>file.maxnum</td>
<td>INT</td>
<td>200</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>file.prstat</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>tran.maxnum</td>
<td>INT</td>
<td>100</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>tran.vertabsize</td>
<td>INT</td>
<td>50000</td>
<td>50</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>audit.log.numbuf</td>
<td>INT</td>
<td>5</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>audit.log.bufsize</td>
<td>INT</td>
<td>1024</td>
<td>0</td>
<td>10 * 1024</td>
</tr>
<tr>
<td>audit.log.maxflsize</td>
<td>INT</td>
<td>1000 * 1024</td>
<td>0</td>
<td>MAX_FL_SZ</td>
</tr>
<tr>
<td>audit.log.syncio</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>restore.log.numbuf</td>
<td>INT</td>
<td>5</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>restore.log.bufsize</td>
<td>INT</td>
<td>1024</td>
<td>0</td>
<td>10 * 1024</td>
</tr>
<tr>
<td>restore.log.maxflsize</td>
<td>INT</td>
<td>1000 * 1024</td>
<td>0</td>
<td>MAX_FL_SZ</td>
</tr>
<tr>
<td>restore.log.syncio</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>recovery.log.numbuf</td>
<td>INT</td>
<td>1000</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>recovery.log.bufsize</td>
<td>INT</td>
<td>1024</td>
<td>0</td>
<td>10 * 1024</td>
</tr>
<tr>
<td>recovery.checkpoint.minsize</td>
<td>INT</td>
<td>100 * 1024</td>
<td>0</td>
<td>MAX_FL_SZ / 2</td>
</tr>
<tr>
<td>recovery.checkpoint.minpercent</td>
<td>INT</td>
<td>20</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>recovery.checkpoint.minreclaim</td>
<td>INT</td>
<td>10 * 1024</td>
<td>0</td>
<td>MAX_FL_SZ / 4</td>
</tr>
<tr>
<td>recovery.checkpoint.mindskfree</td>
<td>INT</td>
<td>50000 * 1024</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>recovery.log.syncio</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>selinux.enforce.force</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>messages.events.selinux.logrow</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>messages.events.selinux.logmsg</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>messages.events.loglevel</td>
<td>INT</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>messages.errors.verbose</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>dispatcher.listenport</td>
<td>INT</td>
<td>9000</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>dispatcher.queuebacklog</td>
<td>INT</td>
<td>150</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>dispatcher.server.idletimeout</td>
<td>INT</td>
<td>5</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>mserver.service</td>
<td>BOL</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mserver.idletimeout</td>
<td>INT</td>
<td>20</td>
<td>-1</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>mserver.fldestroy.service</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mserver.fldestroy.period</td>
<td>INT</td>
<td>3600</td>
<td>-1</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>mserver.bufflush.service</td>
<td>BOL</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mserver.bufflush.period</td>
<td>INT</td>
<td>5</td>
<td>0</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>mserver.bufflush.highratio</td>
<td>INT</td>
<td>70</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>mserver.bufflush.lowratio</td>
<td>INT</td>
<td>30</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>audit.log.path</td>
<td>STR</td>
<td>/var/lib/RUBIXdbms/databases</td>
<td>1</td>
<td>MAX_PATH_LEN</td>
</tr>
<tr>
<td>restore.log.path</td>
<td>STR</td>
<td>/var/lib/RUBIXdbms/databases</td>
<td>1</td>
<td>MAX_PATH_LEN</td>
</tr>
<tr>
<td>recovery.log.path</td>
<td>STR</td>
<td>/var/lib/RUBIXdbms/databases</td>
<td>1</td>
<td>MAX_PATH_LEN</td>
</tr>
<tr>
<td>lob.datapath</td>
<td>STR</td>
<td>/var/lib/RUBIXdbms/databases</td>
<td>1</td>
<td>MAX_PATH_LEN</td>
</tr>
<tr>
<td>volume.datapath</td>
<td>STR</td>
<td>/var/lib/RUBIXdbms/databases</td>
<td>1</td>
<td>MAX_PATH_LEN</td>
</tr>
<tr>
<td>lob.maxopen</td>
<td>INT</td>
<td>500</td>
<td>1</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>

A detailed description of the effects of each configuration option follows:
→ **ipc.lcktimeout**  
The timeout in seconds for an IPC buffer or file lock. If the lock is not satisfied within the number of seconds the lock attempt fails. The timeout prevents deadlocks in a server process when an error condition in another server process results in a lock not being released. This value should be large enough to accommodate any lock wait that occurs due to normal database processing. It should be small enough to produce an acceptable wait when another server fails to release an IPC lock.

→ **ipc.ismshm**  
A flag indicating if the TR Server should use Intimate Shared Memory for the Common Server shared memory segment. Intimate Shared Memory results in faster performance by sharing the virtual memory maps between all processes attached to the shared memory. It also attempts to lock the shared memory into the physical memory resulting in no disk swapping. This value should be set to TRUE unless the platform hosting the TR Server also supports many other services that require large amounts of main memory.

→ **buffer.numbuf**  
The number of database page sized buffers in the buffer manager maintained in shared memory. This value should be large enough to accommodate any database work load anticipated. If the buffer manager hit ratio is unsatisfactory, increasing this value may remedy the problem. Setting this value too large would waste main memory resources or cause undo virtual memory swapping.

→ **buffer.prtstat**  
A flag indicating if the buffer manager should print statistics in the error log upon termination. The statistics are valuable to determine the performance behavior of the buffer manager and to determine if the related configuration options are set correctly for a given database workload.

→ **file.maxnum**  
The maximum number of Trusted Rubix files that may be opened at any time. This corresponds to the size of the main memory file open table. Each entry in the file open table is the size of one database page. In general, each table and index corresponds to a single TR file. Setting this value too small would result in transaction failures. Setting this value too large would result in wasted main memory resources or undo virtual memory swapping.

→ **file.prtstat**  
A flag indicating if the file manager should print statistics in the error log upon termination. The statistics are valuable to determine the performance behavior of the file manager and to determine if the related configuration options are set correctly for a given database workload.

→ **tran.maxnum**  
The maximum number of concurrently active transactions. This corresponds to the size of the transaction table that is maintained in shared memory. Setting this value too small would result in transaction failures. Setting this value too large would result in wasted main memory resources or undo virtual memory swapping.

→ **tran.vertabsze**  
The size of the in-memory version table. The version table is used to map read/write requests to the proper version of the object be accessed. Setting this value too small will result in transactions which perform a large number of writes failing. Setting this value too large will result in an inefficient use of main memory. The version table size should be set to approximately twice the maximum number of write operations (INSERT, DELETE, or UPDATE) of any transaction. If the version table is too small, an error
message will be produced in the error log file.

→ **audit.log.numbuf**
   The number of shared memory buffers used to hold audit log records prior to writing them to disk. This value must be large enough to hold the largest audit record produced by any database operation. In general, it should be large enough such that the buffers are flushed only when a transaction commits. The audit buffers will be flushed when a transaction commits or when a log record is generated by a database operation and there is not enough free space in the audit buffers to hold it. Setting this value too small would result in transaction failures or undo audit buffer flushing. Setting this value too large would result in wasted main memory resources or undo virtual memory swapping.

→ **audit.log.bufsize**
   The byte size of one audit buffer. Audit buffer disk IO is performed in buffer sized amounts. Smaller values tend to increase the number of system calls for disk IO but reduce the number of empty bytes written. Empty bytes may be written when a transaction commit forces its audit records to be written to disk, but the buffer containing those records has not been completely filled. Larger values tend to reduce the number of system calls for disk IO but increase the number of empty bytes written.

→ **audit.log.maxflsize**
   The maximum size of an audit log file. When an audit log file reaches this size a new one will be created. Smaller values result in a finer grain of control when deleting unneeded audit log files, but produce a larger number of files. Larger values result in a lesser grain of control when deleting unneeded audit log files, but produce a smaller number of files.

→ **audit.log.syncio**
   A flag indicating if synchronous IO should be performed when writing audit buffers to disk. Synchronous IO writes the data to the underlying hardware device instead of just the operating system’s buffers. Setting this flag to TRUE would guarantee that the audit log record would be available even if the operating system crashed immediately after a transaction committed; however, it results in having to wait for the IO to be performed when the transaction commits. Setting this flag to FALSE would result in increased performance because the audit records would just be copied into the operating system’s buffers when the transaction commits; however, this would result in a potential loss of the audit record if the operating system were to crash before it wrote its buffers to the physical disk.

→ **restore.log.numbuf**
   The number of shared memory buffers used to hold restore log records prior to writing them to disk. This value must be large enough to hold the largest restore record produced by any database operation. In general, it should be large enough such that the buffers are flushed only when a transaction commits. The restore buffers will be flushed when a transaction commits or when a log record is generated by a database operation and there is not enough free space in the audit buffers to hold it. Setting this value too small would result in transaction failures or undo restore buffer flushing. Setting this value too large would result in wasted main memory resources or undo virtual memory swapping.

→ **restore.log.bufsize**
   The byte size of one restore buffer. Restore buffer disk IO is performed in buffer sized amounts. Smaller values tend to increase the number of system calls for disk IO, but reduce the number of empty bytes written. Empty bytes may be written when a transaction commit forces its restore records to be written to disk, but the buffer containing those records has not been completely filled. Larger values tend to reduce the number of system calls for disk IO, but increase the number of empty bytes written.

→ **restore.log.maxflsize**
The maximum size of a restore log file. When a restore log file reaches this size a new one will be created. Smaller values result in a finer grain of control when deleting unneeded restore log file, but produce a larger number of files. Larger values result in a lesser grain of control when deleting unneeded restore log files, but produce a smaller number of files.

→ **restore.log.syncio**
A flag indicating if synchronous IO should be performed when writing restore buffers to disk. Synchronous IO writes the data to the underlying hardware device instead of just the operating system’s buffers. Setting this flag to TRUE would guarantee that the database would be able to perform restore even if the operating system crashed; however, it results in having to wait for the IO to be performed when the transaction commits. Setting this flag to FALSE would result in increased performance because the restore records would just be copied into the operating system’s buffers when the transaction commits; however, this would result in a potential for the system to not be able to be restored if the operating system crashed. In both cases the database would be able to be restored if only the TR server crashed.

→ **recovery.log.numbuf**
The number of shared memory buffers used to hold recovery log records prior to writing them to disk. This value must be large enough to hold the largest recovery record produced by any database operation. In general, it should be large enough such that the buffers will hold all recovery records that belong to a single transaction. This would ensure that recovery log records will be flushed only when a transaction commits and that if a transaction is rolled back, all recovery records may be read without doing disk IO. The recovery buffers will be flushed when a transaction commits or when a log record is generated by a database operation and there is not enough free space in the recovery buffers to hold it. When a transaction is rolled back all of its recovery records are read to undo the effects of the transaction. Setting this value too small would result in transaction failures, undo recovery buffer flushing, or large amounts of disk IO reading in recovery log records when a transaction rollsback. Setting this value too large would result in wasted main memory resources or undo virtual memory swapping.

→ **recovery.log.bufsize**
The byte size of one recovery buffer. Recovery buffer disk IO is performed in buffer sized amounts. Smaller values tend to increase the number of system calls for disk IO, but reduce the number of empty bytes written. Empty bytes may be written when a transaction commit forces its recovery records to be written to disk, but the buffer containing those records has not been completely filled. Larger values tend to reduce the number of system calls for disk IO, but increase the number of empty bytes written.

→ **recovery.log.syncio**
A flag indicating if synchronous IO should be performed when writing recovery buffers to disk. Synchronous IO writes the data to the underlying hardware device instead of just the operating system’s buffers. Setting this flag to TRUE would guarantee that the database would be able to perform restore even if the operating system crashed; however, it results in having to wait for the IO to be performed when the transaction commits. Setting this flag to FALSE would result in increased performance because the recovery records would just be copied into the operating system’s buffers when the transaction commits; however, this would result in a potential for the system to not be able to be recovered if the operating system crashed. In both cases, the database would be able to be recovered if only the TR server crashed. If this flag is set to FALSE the restore.log.syncio flag should be set to TRUE; otherwise, the database may be corrupted beyond repair if the operating system crashes.
→ **recovery.checkpoint.minsize**
   The minimum size of the recovery log file prior to a checkpoint being performed.
   Checkpointing synchronizes the shared memory buffers with the disk and removes as much of the recovery log file as possible. This must be performed periodically to ensure that the recovery log does not use up so much disk space that the database becomes unrecoverable. Setting this to a smaller size will result in more checkpoints during normal database processing, but less work if a recovery is actually performed following a crash. Setting this to a larger value will result in fewer checkpoint during normal database processing, but more work if a recovery is actually performed. If the recovery log file is allowed to grow too large transactions may fail in order to allow a checkpoint to be performed prior to free disk space becoming too small.

→ **recovery.checkpoint.minpercent**
   The minimum percentage of log file size over the recovery.checkpoint.minsize before starting a complex checkpoint. There are two kind of checkpoint, simple checkpoint and complex checkpoint. Simple checkpoint synchronizes the shared memory buffers with the disk and removes all the recovery log file; complex checkpoint synchronizes the shared memory buffers with the disk and removes as much of the recovery log file as possible, except the recovery log that holds records for active transactions will not be removed and it will be moved to a new recovery log file.

→ **recovery.checkpoint.minreclaim**
   The minimum number of bytes that will be reclaimed during a checkpoint. Checkpointing synchronizes the shared memory buffers with the disk and removes as much of the recovery log file as possible. The amount of the recovery log file that holds records for active transactions will not be removed. This value sets the difference between the size of the recovery log file and those bytes which will not be removed. Larger values will result in fewer checkpoints being performed and more space being reclaimed per checkpoint. Setting this value too large may result in too few checkpoints being performed. Smaller values will result in more checkpoints being performed with less space reclaimed per checkpoint.

→ **recovery.checkpoint.mindskfree**
   The minimum disk free space that need to be available in the system prior to a simple checkpoint or complex checkpoint being performed; if there is no enough free disk space then a panic checkpoint will be performed. The system needs to have enough free disk space for recovery if the system fails while it is doing the checkpoint.

→ **selinux.enforce.force** (SELinux only)
   Force the SELinux policy enforcement within Trusted RUBIX despite the setting for the operating system as a whole.

→ **messages.events.selinux.logrow** (SELinux only)
   Log the SELinux denial of selecting a row. Because the denial of selecting a row may be a normal operation and may happen often, and thus produce a large amount of messages, this functionality of off by default and may be optionally turned on by the administrator.

→ **messages.events.selinux.logmsg** (SELinux only)
   Log SELinux policy denials of Trusted RUBIX objects. Turning on this functionality may be very useful when debugging custom SELinux RDBMS policy.

→ **messages.events.loglevel**
   The level of event logging that is produced by the Trusted RUBIX server and commands. The integer value produces an increased level of logging as the value increases. Higher levels of logging include the lower levels as well. Possible values are: 0) no event logging; 1) log the startup/shutdown of processes as well as any authentication; 2) log RDBMS session connect/disconnect and recovery check-pointing; 3) log client
operations received by the Trusted RUBIX dispatcher and server (no row data is logged for SQL operations); 4) log verbose Trusted RUBIX dispatcher operations and startup process security attributes; 5) log row data for SQL operations; 6) log socket communications.

→ **messages.errors.verbose**
   When set to TRUE verbose error messages are produced when a Trusted RUBIX server or command encounters an error.

→ **dispatcher.listenport**
   The socket communications port dedicated to the TR Dispatcher.

→ **dispatcher.queuebacklog**
   The socket communications queue backlog value to be used for client to dispatcher communications. This value should be larger than the maximum number of concurrent client applications.

→ **dispatcher.server.idletimeout**
   The number of seconds after which an idle server process will be terminated by the dispatcher. The dispatcher maintains a pool of idle server processes to speed up client connection requests.

→ **mserver.service**
   When set to ON the Maintenance Server for a particular database will automatically start when a server process for that database is started. When set to OFF the Maintenance Server will not automatically start. In either case, it may be explicitly started using the rxdb command (see Administrative Commands Reference Guide).

→ **mserver.idletimeout**
   The number of seconds of idle time after which the Maintenance Server will exit. Specifying a value of -1 will cause the Maintenance Server to perform all maintenance services one time and exit. Specifying a value of 0 will cause the Maintenance Server to never idle timeout, in which case it must be terminated using the rxdb command (see Administrative Commands Reference Guide).

→ **mserver.fddestroy.service**
   When set to ON the file destroy service of the Maintenance Server will be periodically performed (assuming the Maintenance Server itself is enabled). When set to OFF the file destroy service will not be performed. The file destroy service physically deallocates any database files that correspond to dropped database objects (e.g., tables, schemata). Only database files that correspond to database objects that will never be used by any future transactions are destroyed.

→ **mserver.fddestroy.period**
   The number of seconds between executions of the file destroy service. A value of -1 will cause the file destroy service to be executed only one time during the execution of the Maintenance Server.

→ **mserver.bufflush.service**
   When set to ON the buffer flush service of the Maintenance Server will be periodically performed (assuming the Maintenance Server itself is enabled). When set to OFF the file buffer flush service will not be performed. The buffer flush service flushes dirty buffer pages to persistent storage (e.g., the disk). Performing this service in the background may increase performance by preventing user submitted database operations from waiting for clean buffer pages. This is especially true when there is a high rate of database writes.

→ **mserver.bufflush.period**
   The number of seconds between executions of the buffer flush service.
→ **mserver.bufflush.highratio**
   The ratio of clean buffer pages to total buffer pages, expressed as a percentage, at which the buffer flushing service will stop flushing dirty pages. For example, if the value is 70 then the buffer flush service will stop flushing dirty pages when 70% (or more) of the pages in the buffer pool are clean.

→ **mserver.bufflush.lowratio**
   The ratio of clean buffer pages to total buffer pages, expressed as a percentage, at which the buffer flushing service will start flushing dirty pages. For example, if the value is 20 then the buffer flush service will start flushing dirty pages when at 20% (or less) of the pages in the buffer pool are clean.

→ **audit.log.path**
   The audit log storage directory for newly created databases. The directory must exist and be writable by the `rubix:rubixtp` user/group. To move the storage directory after a database has been created please see the `rxlogs` command in the Trusted RUBIX Administrative Commands reference Guide.

→ **restore.log.path**
   The restore log storage directory for newly created databases. The directory must exist and be writable by the `rubix:rubixtp` user/group. To move the storage directory after a database has been created please see the `rxlogs` command in the Trusted RUBIX Administrative Commands reference Guide.

→ **recovery.log.path**
   The recovery log storage directory for newly created databases. The directory must exist and be writable by the `rubix:rubixtp` user/group. To move the storage directory after a database has been created please see the `rxlogs` command in the Trusted RUBIX Administrative Commands reference Guide.

→ **lob.datapath**
   The LOB storage directory for newly created databases. The directory must exist and be writable by the `rubix:rubixtp` user/group. To move the storage directory after a database has been created please see the `rxdb` command in the Trusted RUBIX Administrative Commands reference Guide.

→ **lob.maxopen**
   The maximum number of LOB objects that may be accessed an any given time.

→ **volume.datapath**
   The volume (table data) storage directory for newly created databases. The directory must exist and be writable by the `rubix:rubixtp` user/group. To move the storage directory after a database has been created please see the `rxdb` command in the Trusted RUBIX Administrative Commands reference Guide.
INDEX

A
Abstract machine testing, 20
Access control list (ACL)
  audit information, 70
Access Control List (ACL), 45, 71
  and objects, 26
Administrative events audit information, 71
Administrative functions, 49, 62
Administrative roles, 22, 37
  assurance objectives, 23
  audit administrator, 44
  authorizations, 37
  database administrator, 44
  default roles defined, 4
  editing, 50, 52
  operator, 45
  security administrator, 45
Solaris 8 role accounts, 50
Solaris management console, 52
user authorizations, 48
Alter session set label
  and cross domain transactions, 20
audit event, 68
  AUTHORIZATIONS, 39
example, 46, 47
ALTER TABLE

AUDIT EVENTS, 67
  audit information, 70
  AUTHORIZATIONS, 41

Assurance, 32
  and administrative roles, 23
  and auditing subsystem, 72
  architecture, 9, 11
  EAL, defined, 3
  Trusted Operating System integration, 32
Audit, 64
  audit modification event, 68
  audit review audit event, 68
  common tasks example, 76
  data maintenance, 78
Audit administration, 64
  audit events. See Audit event commands, 71
data, 68
  log familiarization, 75
  post-selection, 74
  pre-selection, 72
  rxauditrpt, 72, 74
  rxauditset, 72, 73
Audit administrator. See administrative roles
defined, 4
Audit data, 68

© 2014 Infosystems Technology, Inc.
Audit event, 64

ACL modification, 68
administrative events, 68
and User/Group ID, 17
catalog events, 66
cursor close, 68
cursor open, 68
database events, 66
defined view events, 66
fetch, 68
index events, 67
miscellaneous, 68
relation events, 66
row events, 67
schema events, 66
setting, 73
transaction events, 67

Audit information

administrative events, 71
catalog events, 69
common, 69
database events, 69
defined view events, 69
index events, 70
miscellaneous events, 70
relation events, 70
row events, 70
schema events, 69
transaction events, 70

Audit log familiarization, 75

Authentication, 7, 23
Authorization, 37
and administration, 43
and administrative roles, 24
and audit, 29, 41
and DAC, 40
and database administrator, 44
and MAC, 38
and operator, 45
and rights profiles, 49
and roles, 37
and security administrator, 45
and Trusted Operating System, 16
and Trusted Operating System, 13
audit administrator, 44
backup, 42, 57, 59
DAC bypass, 27
defined, 49
file entries, 50
for specific database, 38
granting, 48
implied, 40
managing, 52
media restore, 56
reclassification, 62
restore, 42, 56, 61
role based access control, 16
user, 43, 48
warning, 40
B

Backup, 31, 57
  auditing, 68
  authorizations, 42, 45
  default role assignment, 37, 45
  full, 31, 59
  latest consistent moment (LCM), 58
  log book, 59
  quiescence time, 57
  restore logging, 59
  security-related, 35
  steps, 59
  warning, 58
Backup database, 42, 57, See rxdump
  audit event, 66
  audit information, 69
Bulk loading, 54, See rximport command
  different label, 55
  same label, 54

C

Catalog events audit information, 69
  audit event, 66
  audit information, 70
Commit transaction
  audit event, 67
  to close a transaction, 47
Common criteria administrative roles requirement, 24
  audit event, 66
  audit information, 69
Compartment
  in classification, 2
Configuration management, 33
  audit information, 70
Cover story, 25
  audit information, 70
Cover channel
  in polyinstantiation, 28
  audit information, 70

© 2014 Infosystems Technology, Inc.
D

DAC security policy, 14, 25. See authorizations and DAC

ACL, 26

GRANT NULL PRIVILEGE, 26

listing user DAC privileges, 24

NULL PRIVILEGE, 26

Database

administering authorizations, 16

administrative authorizations, 43

and Trusted Operating System, 7

architecture, 6

assurance, 32

audit administration, 64, 71

audit authorizations, 41

audit events, 65

authorization effectiveness, 16

backup, 31, 57

backup log book, 59

backup steps, 59

bulk loading, 54

create database, 28

data security policy, 25

default roles, 37, 44

defined, 2

delivery and installation, 79

full backup, 59

identification and authentication, 13

listing user DAC privileges, 24

logging, 30, 56

MAC authorizations, 38

major subsystems, 9

media restore, 31, 56

multilevel objects, 15

polyinstantiation, 28

recovery, 30, 55

restore, 31, 60

restore authorizations, 42

restore logging, 30, 59

roll forward, 32, 61

row reclassification, 62

security-relevant events, 35

session sensitivity label, 14

system recovery, 30, 56

transaction recovery, 30, 56

Trusted Operating System dependencies, 12

trusted recovery, 30

user authorizations, 43

user management, 22

verified installation, 80

Database administrator. See administrative roles

defined, 4

database events audit information, 69

Database operator

defined, 4

Database Storage Locations, 63

Databases directory, 19

Default roles, 16. See administrative roles

defined, 4

operations matrix, 48

Delete row
audit event, 67
audit information, 70

Delivery and installation, 79

Discretionary Access Control (DAC), 25
  authorizations, 40
  exempt, 45
  null privilege, 23

Dispatcher, 7
  configure, 88

Documentation, 36

Drop
  DAC authorization, 41

Drop catalog
  audit event, 66
  audit information, 69

Drop database
  audit event, 66
  audit information, 69

Drop index
  audit event, 67
  audit information, 70

Drop schema
  audit event, 66
  audit information, 69

Drop table
  audit event, 66
  audit information, 70

Drop view
  audit event, 66
  audit information, 69

© 2014 Infosystems Technology, Inc.
audit event, 67
DAC authorization, 40
Installation
verified, 80
Internal transfer of data, 19

L
Latest Consistent Moment (LCM), 58, 59, 60
Least privilege
and RBAC, 16
and Trusted Operating System, 12
defined, 3
Life cycle support, 34
Logbook, 58, 59, 61
Logging, 30, 56
and restore, 32
archiving and LCM, 59
basis, 60
changing file size, 60
file name, 59
media recover, 56
recovery, 60
restore logging, 59
shadow directory, 60
system recovery, 56
transaction recovery, 56
types, 30
warning, 57
Login
account types, 49
and clearance sensitivity label, 13

M
MAC security policy, 23, 25
Maintenance server, 7
configure, 88
Mandatory Access Control (MAC), 27
and kernel subsystem, 9
and object reuse, 18
and polyinstantiation, 28
and session sensitivity label, 22
and Trusted Operating System, 19, 46
and Trusted Operating System, 19
and Trusted Operating System, 15
and user security policy, 23
at object creation, 28
authorizations, 38, 50
defined, 3
enable bypassing, 62
polyinstantiation, 23
security lattice, 15, 27
security policy, 15, 23, 25
Media failure, 31, 55
Media restore, 56, See rxrestore
authorization, 43
responsibility, 31, 56
Miscellaneous events audit information, 70
Multilevel data import
authorization, 43, 46
warning, 15
Multilevel secure, 13
defined, 3
Multi-version Timestamp Ordering (MVTO), 18

N
Named object
   protection, 15
Named objects
   and DAC, 26
Notes, 47

O
Object owner, 25
Object reuse, 17
   and Trusted Operating System, 18
MVTO, 18
Open database
   audit event, 66
   audit information, 69
Operations
   alter session set label, 39, 46, 47
   audit, 64
   audit report, 41, 74
   audit set, 41, 72, 73
   backup database, 42
   DAC exemption, 40
   data export. See rxexport
data import. See rximport
database backup, 57, See rxdump
database recovery, 55, See rxrestore
delete audit log files, 42
drop database, 43, 62
export data, 44
import data, 44
list audit log files, 42
list database, 43, 62
list restore log files, 42
maintenance server shutdown, 43
maintenance server startup, 43
move (rename) database, 43, 62
restore database, 43
set audit log directory/files, 42
set restore log directory/files, 42
Operator. See administrative roles

P
Polyinstantiation, 23, 25, 28, 62
Post-selection criteria, 74, See rxauditrt
Pre-selection audit criteria, 72, See rxauditset
Process
   client, 7
   dispatcher (rxdspr), 7
   maintenance server (rxmsvr), 7
   server (rxserver), 7
Process architecture, 6
   warning, 6

Q
Quiescence time, 57

R
Recovery, 55, See rxrestore
   file protection, 15
trusted, 30

Trusted Operating System recovery from failure, 20

Reference monitor, 18

Relation events audit information, 70

Residual information protection, 18

Restore database, 32, 42, 60, See rxrestore

Restore logging, 59

Revoke

audit event. See grant, audit event

DAC authorization, 41

Rights profile

defined, 49

editing, 50

execution shells, 50

managing through GUI, 52

Role account, 49

Role Based Access Control (RBAC), 16, 49

vs. role account, 50

Roles

combining authorizations, 16

default, 44

editing, 51

Roll forward, 32, 61

Rollback

audit event, 67

log, 56

to close a transaction, 47

transaction, 30

Rollback transaction

audit event, 67

Row events audit information, 70

Row polyinstantiation, 28

Row reclassification, 38, 39, 62

audit event, 68

RUBIX logon, 13

RUBIX USER, 19

RUBIX/RUBIXTP, 19

RUBIXTP group, 19

rxauditrpt, 71

audit event, 68

authorization, 41

example, 74

rxauditset, 71

audit event, 68

authorization, 41

example, 73

rxconfig file, 7, 10, 82

rxdb, 62

audit event, 68

drop database authorization, 43

list database authorization, 43

move database authorization, 43

rxdspr, 7

rxdump, 59

backup database authorization, 42

rxexport

and cross domain transactions, 20

audit event, 68

export database authorization, 44

warning, 15
rximport

and cross domain transactions, 19
audit event, 68
import database authorization, 44
warning, 15

rxlogs, 60
audit event, 68
delete audit logs authorization, 42
delete restore logs authorization, 42
list audit logs authorization, 42
list restore logs authorization, 42
set audit log directory authorization, 42
set restore logs directory authorization, 42

rxmsvr, 7

rxrestore, 32, 56, 61
audit event, 68
restore database authorization, 43

rxserver, 7

rxsvrman, 7, 14, 43

S

Schema events audit information, 69

Security administrator. See administrative roles
defined, 4
role creation, 37
Trusted Operating System duties, 15
warning, 15

Security audit, 17, 29
Security Features User Guide (SFUG), 34
Security function protection, 18
Security lattice, 14

Security management console, 52
and rubix.*.grant authorization, 38
and security rights profile, 51
eexample, 52

Security Policy. See DAC security policy, See MAC security policy
and administrative roles, 23
data, 25
defined, 4
enable bypassing, 62
security audit, 17, 30
user, 22

Select row

AND CURSOR CREATION, 65
and sql_cursor_close audit event, 65
and sql_cursor_open audit event, 65
cursor close audit event, 68
cursor open audit event, 68
DAC authorization, 40
fetch audit event, 68
fetch audit information, 70

Session label
altering, 46
and importing data, 44
and insert, 47
and MAC decisions, 14
authorizations, 38
relationship to system label, 46
reset, 47

Session sensitivity label, 13
maximum, 13
### Trusted Facility Manual (TFM) documentation, 35

- Set-gid, 49
- Set-uid, 49
- Solaris. See Trusted Solaris 8
- Subject binding, 13
- System label. See session label relationship to session label, 46
- System recovery, 31, 56

**T**

- Transaction events audit information, 70
- Transaction failure and recovery, 30
- Transaction logging, 60, See \textit{xrestore}
- Transaction recovery, 30, 56
- Trusted Computing Base (TCB), 13, 64
  - and common server subsystem, 9
  - and named system objects, 15
  - and object reuse, 18
  - and protection mechanisms, 19
  - assurance, 32
- Trusted Operating System, 1, 7
- and common server subsystem, 9
- and named system objects, 15
- and object reuse, 18
- and protection mechanisms, 19
- authentication, 13
- authorizations, 16
- dependencies, 12
- identification and authentication, 13
- MAC relationship, 15
- recovery from failure, 20
- security administrator duties, 15
- security database files, 50
- security lattice, 27
- subject binding, 13
- user base coordination, 35
- warning, 6

**U**

- Trusted recovery, 30
- Trusted RUBIX server configuration, 82
- Trusted RUBIX Dispatcher operation, 10

- Update row audit event, 67
- audit information, 70
- DAC authorization, 40
- reclassify row, 62
- User

© 2014 Infosystems Technology, Inc.
administrative, 23
normal, 23
security attribute definition, 14
subject binding, 14
User authorizations. See administrative roles
User ID
and ACL, 26, 71
and DAC, 14, 22
audit, 69
effective, 49, 51
from Trusted Operating System, 7, 49
View events audit information, 69
Warning, 6, 15, 21, 22, 57, 58, 61, 80
Write-Ahead Logging, 30