Abstract: A multiuser relational DBMS system, Trusted RUBIX, has been designed and implemented to satisfy the requirements of the TCSEC at the B2 class. In this paper, the architecture of Trusted RUBIX is presented, its integration within the B2 UNIX System V platform is discussed, and the adaptation and interpretation of the SeaViews security policy model in Trusted RUBIX are explained. The lessons learned from this design and implementation exercise are also discussed.

1: Introduction

The effective use of trusted operating system technology in the marketplace can only be achieved if common trusted applications such as database management systems and message systems become widely available. Few DBMSs that are designed to satisfy the trust requirements of evaluation class B2 have been developed to date even though both trusted operating systems [ATT, TRXE] and guidelines for the evaluation of these DBMSs [TDI] have been available. Part of the reason for the dearth of trusted DBMS is that their integration with trusted operating systems poses unanticipated design and implementation challenges.

This paper discusses the structure of the Trusted RUBIX relational DBMS, which has been implemented to run on standard B2-class UNIX systems. The goals set forth for the development of Trusted RUBIX are first presented. The product architecture is then described, and the rationale for various design choices is presented. The adaptation and subsequent interpretation of the SeaViews security policy model within Trusted RUBIX are defined. The paper concludes with a discussion of the lessons learned from this design, which we believe are applicable to other trusted DBMSs.

2: Trusted RUBIX goals and architecture overview

Trusted RUBIX is a full-featured relational database management system intended to provide (B2) trust when layered upon a POSIX-compliant environment. The system includes a complete set of data management tools; a user-friendly, forms-based interface; ANSI-compliant SQL (both embedded and interpretive); and a procedural report-writer language. The DBMS further provides an object-oriented C-language application library, as well as a set of command tools which support a powerful relational algebraic syntax and are seamlessly integrated into the UNIX shell programming environment.

2.1: Goals

The principal goal of the Trusted RUBIX design was the use of a standard operating system platform evaluated at class B2. This goal was deemed important both to ensure the market exposure of Trusted RUBIX and to facilitate its evaluation, e.g., by using the...
security policies and mechanisms of the underlying operating system. A second goal of the design was that of maintaining compatibility with existing standards and standard interfaces (e.g., ANSI SQL compliance). This goal is important because of the large number of already available DBMS applications that require compliance with ANSI SQL interface standards. A third goal of the Trusted RUBIX design was the use of a general, well-understood and accepted security policy model. This goal is also critical in view of the fact that support of a well-defined policy is one of the two key security tenets, the other being that of penetration resistance. The fourth design goal was that of retaining the superior performance characteristics of the RUBIX RDBMS in environments of small multiuser systems. Although superior performance is a generic goal of all designs, in small system environments it is especially important due to the unavailability of a large set of architecture options to enhance overall system performance.

In order to achieve the above goals, the operating system platform chosen was UNIX System Laboratories' B2 Trusted UNIX System V Release 4ES. (However, other UNIX-based platforms, such as Trusted Xenix, could be used with Trusted RUBIX without significant porting effort.) The SeaViews security policy model was chosen as the Trusted RUBIX model and adapted to the Trusted RUBIX environment. These adaptations are described in the ensuing sections.

2.2: Architecture overview

The combination of design goals set forth above for all practical purposes rules out the allocation of a trusted computing base (TCB) subset [TDI] to the Trusted RUBIX DBMS outside of the B2 UNIX TCB. For example, consider the implications of encapsulating Trusted RUBIX in a protected subsystem outside of the UNIX TCB, which (1) would be trusted by its user applications and (2) would be untrusted with respect to the underlying UNIX TCB. Although such a protected subsystem would be able to rely on the UNIX TCB policies and mechanisms (e.g., identification and authentication, mandatory access control, discretionary access control, audit), it would require that different RUBIX tuples or views of the same or different relations be stored in separate files at different levels whenever these tuples or views have different levels.

The use of objects of the underlying TCB subset to represent objects of the DBMS subset, which was first advocated by Hinke and Schaefer [HS], has the following disadvantages in the context of our design goals:

(a) Inflexibility of DBMS subset policy

Using a Hinke-Schaefer-like architecture, there is no choice in the polyinstantiation discipline used when a LOW subject inserts a tuple that duplicates a preexistent HIGH key. This is the case because the DBMS code, running at LOW, is unable to make any decisions that depend upon the presence or absence of HIGH tuples in the database. By contrast, using the trusted subject architecture, a choice of polyinstantiation disciplines can be offered to the user, including the POLYHIGH and POLYLOW policies of [SAJA] or no polyinstantiation at all.

(b) Unresolvable conflict with ANSI DBMS standard features

Using a Hinke-Schaefer-like architecture, ON DELETE RESTRICT cannot be enforced because the constrained LOW subject should not discover the presence of dangling HIGH references. By contrast, using the trusted subject architecture, it is possible to implement the complete ANSI Level I referential integrity semantics. In principle, in a trusted subject architecture, the trusted subject can determine that dangling HIGH references exist for an object being deleted at LOW, and it can upgrade the object instance at HIGH before deleting it at LOW. Thus, the conflicts between the referential integrity constraints and multilevel security can be resolved in some instances.

(c) Potential performance loss

The trusted subject architecture can be expected to outperform a Hinke-Schaefer-like architecture in most environments. This is because most applications are read-intensive and there is less file access overhead in reading from a single operating system file and executing internal logic than in opening multiple files, reading, and interleaving the tuples encountered therein.

A trusted subset architecture is not required for B2 trusted systems but is consistent with such systems [TDI]. B2 UNIX TCBS, support a monolithic operating system kernel, instead of
a classic security kernel, which can be isolated from the rest of the trusted base (e.g., trusted processes) via a ring or domain isolation mechanism. Consequently, the Trusted RUBIX RDBMS can be integrated with the underlying B2 UNIX TCB to form a single TCB.

The integration of the RUBIX and UNIX TCBS is illustrated in Figures 1 and 2. The integration requires that the RUBIX mandatory access control model be composed with the UNIX mandatory access control model. This composition defines the MAC properties of the subjects and objects visible to untrusted RUBIX and UNIX users at the common TCB interface. These MAC properties enable untrusted RUBIX users to access both RUBIX and UNIX objects in a controlled fashion but prevent untrusted UNIX users from accessing RUBIX objects. This is illustrated in Figure 1. The internal structure of Trusted RUBIX servers and of untrusted RUBIX applications (shown in Figure 1) is illustrated in Figure 2.

The integration of the RUBIX and UNIX TCBS, and the composition of their attendant models, relies upon two independent non-discretionary access control mechanisms. The first mechanism is the separation of RUBIX subjects and objects from UNIX subjects and objects via definition of two disjoint sublattices of the UNIX security level lattice. This is achieved by the definition of the RUBIX category and the reservation of an attendant sublattice of the UNIX lattice. Subjects and objects whose security levels belong to the RUBIX sublattice are inaccessible to subjects executing on the UNIX sublattice by virtue of the MAC rules enforced by the underlying UNIX TCB. To achieve this separation, the security levels in the RUBIX sublattice are defined to be incomparable with any security levels belonging to the remaining UNIX sublattice. Each UNIX security level outside the RUBIX sublattice includes the LOGIN category; the corresponding security level in the RUBIX sublattice is formed by deleting the LOGIN category from the level and attaching the RUBIX category in its place. Note that only data objects and trusted RUBIX subjects ever bear security levels defined on the RUBIX sublattice. The sublattice structure reserved for UNIX objects which represent RUBIX data and trusted subjects is depicted in Figure 3.

The second non-discretionary access control mechanism is the use of a special administrative role, namely, the RUBIXTP (for "RUBIX trusted process") role. This role is implemented as a reserved UNIX group. Membership in this group is non-discretionary because it can only be granted or revoked by trusted administrative users. All trusted RUBIX subjects are members of the RUBIXTP role, and are established by invocation of the set-GID UNIX programs that encapsulate the RUBIX data. All UNIX objects in which RUBIX data is stored are accessible only to trusted RUBIX subjects executing in the RUBIXTP role. Note that trusted RUBIX subjects are unable to grant membership in the role to any untrusted subjects.

The representation of the RUBIX untrusted subject is identical to that of the UNIX untrusted subject. A RUBIX untrusted subject is established by completing the UNIX login protocol and is destroyed by terminating the UNIX session. All security properties of UNIX untrusted subjects [ATT] are inherited by RUBIX untrusted subjects. The RUBIX untrusted subject is itself a client of a RUBIX trusted subject, both of which are established by executing a RUBIX command or C interface function. The condition for creation of a RUBIX trusted subject is that its effective primary group ID must match the RUBIXTP role ID. The security level assigned to the RUBIX trusted subject is initially equal to the UNIX security level of its invoker. However, prior to accessing RUBIX objects, the RUBIX trusted subject changes its security level to the corresponding RUBIX level by deleting the UNIX LOGIN category from the UNIX security level and replacing it with the RUBIXTP category (q.v. Figure 4). (Recall that the RUBIX category is relied upon to isolate RUBIX data from 'non-RUBIX data and processes within the UNIX system.) In order to change levels as indicated above, the RUBIX trusted subject must possess the P_Setlevel privilege, which is explicitly acquired immediately before the change of level and dropped immediately thereafter. The activation of RUBIX trusted subjects and their exploitation of the UNIX privilege mechanisms is depicted in Figure 4.

The composition of the Trusted RUBIX and UNIX security policy models is illustrated in Figure 5. The model composition differs from that of the Sea Views model hierarchy in order
to reflect more closely the specific O/S-DBMS relationships in the Trusted RUBIX/UNIX configuration. Whereas the SeaViews model exhibits two layers, with MAC on segments enforced by M(0) and DAC on database objects enforced by M(1), Trusted RUBIX exhibits a ternary stratification: M(0) represents the MAC, I&A, and TCB isolation and incircumventibility mechanisms provided by UNIX; M(l) represents the Trusted RUBIX MAC mechanisms; and M(2) is introduced to represent the Trusted RUBIX DAC mechanisms.

Assurance features of Trusted RUBIX distinguish it from B1-class DBMSs. For example, the design specification and verification requirements, covert channel analysis requirements [ITI], and penetration resistance requirements satisfied by Trusted RUBIX distinguish it from similar architectures that are only suitable for class B1.

3: Interpreting the SeaViews model in Trusted RUBIX

The primary role of the interpretation of a security policy model within a TCB is that of providing an explanation of how the model is actually supported within the TCB. The requirement for an interpretation of the security policy model can be derived from the TCSEC design documentation requirements of class B1 and B2 systems and from the design specification and verification requirements of class B3 and A1 systems [MILL].

3.1: Adaptation of the SeaViews model

The SeaViews model [LDSHS] consists of the following components:
(1) A set of states which are characterized by a set of state-dependent functions. States are defined in terms of subjects, objects, and access privileges that are identified with state-dependent functions. Note that security levels are defined by state-independent functions to reflect the fact that tranquility is an important mandatory property. Despite this fact, however, the security levels play an important part in the definition of state transitions.
(2) A set of commands and command sequences which represent actions that cause state transitions. It should be noted that the SeaViews model (as well, consequently, as its RUBIX adaptation) does not describe specific system commands or command sequences. Instead, it specifies command and command sequence properties that govern allowable state transitions. This choice affects the interpretation definition and correspondence to the model as explained below.
(3) State, command, and command-sequence properties, which represent formal policy statements and additional design restrictions for the model. State properties define the notion of secure states, whereas command and command-sequence properties help to define the notion of secure state transitions. It should be noted that the SeaViews model includes the notion of transition-specific constraint in the command and command-sequence properties.

(4) A set of types and type properties, which help to define the subjects, objects, access privileges, security levels, and various functions. The SeaViews model includes a set of types together with functions defined upon those types and a set of type properties which apply to state-independent functions only.

Two important modifications to the SeaViews model were undertaken. First was the restriction of properties of the SeaViews model by the addition of constraints to reflect cases where Trusted RUBIX formalisms are less general than their SeaViews analogues. For example, the SeaViews tuple class property is constrained to map SeaViews' element-labeling policy to Trusted RUBIX's record-labeling policy. Second, the model was augmented to reflect differences in security policy and available architecture support. For example, three classes of polyinstantiation behavior were modeled, and both an update property and a multi-attribute foreign key property were added. The effective authorization and initial authorization properties were precisely defined, consonant with the semantics of POSIX ACLs and multiple concurrent group membership.

3.2: The model interpretation

A valid interpretation of a security policy model within a TCB is a mapping from the TCB states to the model states which must show that all TCB states correspond to model states. Thus, to show that a policy model interpretation in a TCB is valid, we must define this mapping and must demonstrate that the TCB state transitions
satisfy (or are consistent with) the model's state transitions.

The definition of the mapping consists of selecting each policy model component and delineating the TCB components which are "assigned" to that model component. The TCB components that are "assigned" to model components include subjects, objects, access privileges, security levels, and TCB interface operations (i.e., TCB function calls and commands). The TCB interface operations include the security-relevant rules, which determine the TCB transition properties and constraints.

The demonstration that the Trusted RUBIX TCB state transitions satisfy the adapted SeaViews policy model, viz., that TCB transitions represent (or correspond to) model-state transitions, is summarized by the diagram shown in Figure 6. In essence, the explanation of the correspondence between the RUBIX TCB transitions and the policy-model transitions can be summarized by the statement:

\[
\text{command-seq( mapping( R_j ))} = \text{mapping( TCB_op( R_j ))}
\]

once the mapping is defined.

Note that the SeaViews policy model and its consequent adaptation to Trusted RUBIX include only command and command-sequence properties but not specific command definitions. Thus, individual Trusted RUBIX TCB operations cannot (and, in fact, need not) be mapped to any model command. Rather, the state-transition rules which govern the Trusted RUBIX TCB operations were shown to satisfy the command and command-sequence properties of the model.

To show that the state transition rules of the Trusted RUBIX TCB satisfy the command and command sequence properties of the SeaViews model for the defined interpretation (mapping): (1) The SeaViews properties are restated in terms of their Trusted RUBIX correspondents; and (2) An explanation of why the restated properties hold true in Trusted RUBIX is supplied.

3.3: Establishing the correspondence between the security policy models (an example)

As an example of the establishment of the correspondence between the Trusted RUBIX and SeaViews security policy models, the interpretation of property 20 (Referential Integrity Property) of the SeaViews model within Trusted RUBIX is considered. In Figure 7, Property 20 is first reformulated to capture the Trusted RUBIX security semantics, and is then restated in English. Finally, a logical explanation or justification of the restated property indicates why the property holds in the Trusted RUBIX TCB:

**Explanation**

Assertion (a) holds by the definition of referential integrity in any RDBMS [DATE], multilevel security notwithstanding. Assertion (b) holds for the following reasons: First, alteration of R (e.g., insertion of the row r) requires an observation of Q. Second, to observe Q, the security level of the untrusted RUBIX application must dominate the security level of q. At the same time, the insertion of r into R sets the security level of r to that of the untrusted RUBIX application. Therefore, the security level of r must dominate that of q.

4: Conclusions

The architecture of Trusted RUBIX, its integration within the trusted base of the B2 UNIX SVR4 platform, and the interpretation of the SeaViews model within Trusted RUBIX were discussed.

Three significant lessons were learned from the project which we believe are applicable to a large class of DBMSs and underlying operating systems. First, attempts to allocate the trusted components of a DBMS to a separate TCB subset implemented as a protected subsystem are likely to be unsuccessful unless the DBMS objects and access control policy are refinements of the underlying operating system objects and access control policy. This was not the case in Trusted RUBIX and, therefore, a trusted server architecture was chosen in which a trusted server is activated on a per-client application basis.

Second, the allocation of the trusted DBMS components to the trusted subjects of the underlying operating system can be performed in such a way that the DBMS trusted subjects can operate with the fewest privileges within the operating system TCB. This is important because it facilitates the analysis if the combined DBMS-O/S TCB.
Third, the separation of trusted DBMS components from untrusted ones (e.g., the SQL data definition, manipulation and authorization language components) is possible. This separation helps decrease the size of the combined DBMS-O/S TCB and, thus, it reduces the TCB analysis effort.
5: References


Figure 1. Trusted RUBIX Architecture (Trusted RUBIX Data Isolation)
Figure 2. Trusted RUBIX Architecture
(Allocation of Trusted RUBIX Components to the UNIX(R) TCB)
Figure 3. Reserved Security Levels for UNIX(R) Objects Representing Trusted RUBIX Data
NOTE: (1) Call I = SetLabel (e.g., U.SL <- Rx.SL)
(2) Call J = CreateDB (e.g., SetLabel, Chowner, Link, etc.)

Figure 4. Effective Privileges and Bracketing in Trusted RUBIX
Figure 5. Relationship between the Trusted RUBIX and UNIX(R) Security Models
Conceptual Diagram of the Model Interpretation in RUBIX

Model State $S_i$ \(\xrightarrow{\text{command\_sequence}}\) Model State $S_j$

RxTCB State $R_i$ \(\xrightarrow{\text{mapping}}\) RxTCB State $R_j$

- Meaning of diagram:

\[\text{command\_sequence}(\text{mapping}(R_i)) = \text{mapping}(\text{TCB\_operation}(R_j))\]

- Alternate statement:

For the defined mapping, the TCB state-transition properties (i.e., rules) satisfy the command sequence properties

Figure 6. The Mapping of the RUBIX TCB to the Security Model
Property 20 (Referential Integrity) A command $op(s_1, S, z_1 \ldots z_n \rightarrow s_2)$ satisfies the referential integrity property if and only if $\forall R \in \text{MREAL-IDS}$, where $n = \text{mdegree}(R)$, $\forall T, J \in \mathcal{P}(i \in \mathcal{N}[i \leq n])$, where $(Q, J) = \text{foreign-key-ref}(R, T)$, $\forall c \in \text{CLASSES}$ | $c \geq \text{mrelation-class}(R)$, $\forall r \in \text{mrelation-instance}(s_2, R, c)$:

(a) $\exists k \in T \mid \text{element-value}(r, k) = \text{null} \Rightarrow$

$\exists k \in T \mid \text{element-value}(r, k) \neq \text{null}$; and

(b) $\exists k \in T \mid \text{element-value}(r, k) = \text{null} \land$

$r \notin \text{mrelation-instance}(s_1, R, c)$

$\exists q \in \text{mrelation-instance}(s_1, Q, c) \mid$

$\text{mtuple-part}(r, T) = \text{mtuple-part}(q, J) \land$

$\text{mtuple-class}(r) \geq \text{mtuple-class}(q)$.

Property 20 (Restated in Trusted RUBIX)

Let $R$ be a named relation of degree $n$ and let $Q$ be another named relation of unspecified degree. Let $r$ be a tuple formed by extracting a number of columns from $R$ and let $q$ be a tuple formed by extracting an equal number of columns from $Q$. A Trusted RUBIX operation (i.e., command or function) satisfies the referential integrity property if and only if the following assertions hold for all security levels that dominate the lowest level at which $R$ is visible:

(a) If any key attribute of $r$ is null, then all of its key attributes are null;

(b) Otherwise, if $r$ was added by the Trusted RUBIX operation, then:

(1) There exists a tuple $q$ in $Q$ such that $r = q$, and;

(2) The security level of $r$ dominates that of $q$. 

110