Formal Treatment of a Secure Database

Roger Bishop Jones
ICL Secure Systems,
Eskdale Road, Winnersh, Berks, UK, RG11 5TT.
Phone +44 734 693131, E-mail R.B.Jones@win0109.uucp

Abstract

This paper consists of an annotated Z specification illustrating proposed methods for the verification of a multi level secure database.
# Contents

1 Introduction 3

2 The Specification of Requirements 4

2.1 The Abstract Syntax of the Query Language 4
2.2 The Type of Models of Systems 5
2.3 Critical Requirements 6
2.4 SSQL specification 7
2.5 correctness property 7

3 A Formal Model of a Secure System Architecture 8

3.1 Component Types 8

3.1.1 The DBMS 8
3.1.2 The FILTER 8
3.2 Construction 9
3.3 Requirements on Components 9

3.3.1 Requirements on DBMS 9
3.3.2 Requirements on DBMS 9
3.4 Architecture Correctness Conjectures 10

4 A Model of a Secure Kernel 10
4.1 The Model 10

5 System Correctness Proposition 11

6 Acknowledgements 11
1 Introduction

The example used in this paper is derived from the definitions of the SWORD relational query languages. Radical simplifications have been undertaken to permit a brief exposition of a method suitable for use (subject to some further elaboration) on the full SWORD specifications.

The nature of these radical simplifications are summarised here.

Firstly the abstract syntax of SWORD SQL is simplified as follows:

Value
Only “denote_integer” and “binop” are considered.

Col_spec
Unchanged.

Table_spec
Unchanged.

Col_name
Not used.

From_spec
Only “Table_spec” is considered.

From_name
Not supported.

Set_clause
Not needed.

Tuple_list
Only “all_tuples” is considered.

Insert_list
Not needed.

Classified_value
Only the “Classify” option is considered.

Select_list
Only “select_values” is considered.

Action
Only “select” is considered.

Query
Only the “action” query is addressed.

Secondly the output from the system is modelled simply as a set of classified data values.

section 2 - a specification of critical requirements

An example of a specification of critical requirements for a secure system is given.
section 3 - a formal model of a system architecture

The first stage in the design and implementation of a system meeting the formalised critical requirements is to establish a system architecture which can be shown to guarantee satisfaction of the requirements. Such an architecture is exhibited, followed by a proof that it does indeed ensure conformance to the critical requirements.

2 The Specification of Requirements

A new theory is introduced for the formal material.

```
sml

new theory "mlsdb";
```

The specification of critical requirements avoids detail about the specific operations supported by the system, and we therefore use ‘given sets’ to represent the inputs and outputs.

```

| [String, Data, Class] |
```

We assume that classifications are partially ordered by the relation \( \text{dominates} \).

```

\text{dominates} : \text{Class} \mapsto \text{Class}
```

\[
\forall x, y, z : \text{Class} \quad \bullet \\
\begin{align*}
& x \text{ dominates } x \\
\wedge & (x \text{ dominates } y \wedge y \text{ dominates } x \Rightarrow x = y) \\
\wedge & (x \text{ dominates } y \wedge y \text{ dominates } z \Rightarrow x \text{ dominates } z)
\end{align*}
\]

2.1 The Abstract Syntax of the Query Language

```

Col_spec == seq String

Table_spec == seq String

From_spec == Table_spec

Value ::= denote_integer \((\langle\langle\text{String}\rangle\rangle)\) | denote_binop \((\langle\langle\text{Value} \times \text{Value}\rangle\rangle)\) | denote_class \((\langle\langle\text{String}\rangle\rangle)\) | contents \((\langle\langle\text{Col_spec}\rangle\rangle)\)

Select_list == seq Value
```

2.2 The Type of Models of Systems

The state of the system is modelled as a classified data space.

\[ \text{STATE} == \text{Class} \leftrightarrow \text{Data} \]

An input to the system is a single classification and a query.

\[ \text{IN} == \text{Class} \times \text{Query} \]

Output from a single query may be of mixed class and is therefore modelled in a way similar to the system state. For the purposes of this example information content in the ordering of the output data is therefore ignored.

\[ \text{OUT} == \text{Class} \leftrightarrow \text{Data} \]

A system is modelled as a transition function. This is to be regarded as a single function modelling all the permissible transitions of the system. To permit a non-interference style exposition of the “no flows down” requirement the transition takes complete input histories and yields the corresponding output history.

We also require that the system is a total deterministic function. This is not likely to be the case in fact. This infelicity in the mathematical models is discussed below.

\[ \text{SYSTEM} == \{ \text{tf}: (\text{seq IN} \times \text{STATE}) \rightarrow (\text{STATE} \times \text{seq OUT}) \mid \forall \text{si}:\text{seq IN}; \text{s,s'}:\text{STATE}; \text{so:seq OUT} \mid (\text{si,s}) \mapsto (\text{s'},\text{so}) \in \text{tf} \mid \# \text{si} = \# \text{so} \} \]

A stronger contraint is desirable here reflecting the expectation that the function is obtained by iterating a single step transition function.
2.3 Critical Requirements

We now attempt to capture (in fact define) what it means to say that such a system is secure. The critical requirement is then that the system be secure.

The intended meaning of secure here concerns the nature of the information flows permitted by the system. We have not come to a firm view on the basis of the available material on whether the observations about sanitisation are offered as consequences of the ‘no flows down’ requirement, or whether they are intended as additional constraints. For the purposes of this example sanitisation is not considered.

The formulation below is an “interference style” formulation. To express the flow constraint in this way it is first necessary to define filtering operations on the inputs and outputs from the system.

Two sequences of inputs $si1$ and $si2$ are the same when viewed from a classification $\text{class}$ if they have the same length and, at all the places where either sequence has an input whose classification is dominated by $\text{class}$, the values in the two sequences are identical.

\[
\text{same_ins} : \text{Class} \rightarrow (\text{seq IN} \leftrightarrow \text{seq IN})
\]

\[
\forall \text{class} : \text{Class}; si1, si2 : \text{seq IN} \bullet
\quad (si1,si2) \in \text{same_ins class} \iff
\quad \# si1 = \# si2
\quad \land
\quad (\forall n: \text{dom si1}
\quad \quad | \text{class dominates (first (si1 n))} \lor \text{class dominates (first (si2 n))}
\quad \quad \bullet si1 n = si2 n)
\]

Two sequences of outputs $so1$ and $so2$ are the same when viewed from a classification $\text{class}$ if they have the same length and, at all the places in the sequence, the restrictions of the two output relations to those classifications dominated by $\text{class}$ are identical.

\[
\text{same_outs} : \text{Class} \rightarrow (\text{seq OUT} \leftrightarrow \text{seq OUT})
\]

\[
\forall \text{class} : \text{Class}; so1, so2 : \text{seq OUT} \bullet
\quad (so1,so2) \in \text{same_outs class} \iff
\quad \# so1 = \# so2
\quad \land
\quad (\forall n: \text{dom so1}
\quad \quad | \{c: \text{Class} | \text{class dominates c}\} \triangleleft (so1 n) = \{c: \text{Class} | \text{class dominates c}\} \triangleleft (so2 n))
\]

The constraint on information flows proposed for verification is that outputs classified at classes dominated by any class $\text{c}$ are independent of inputs at classifications which do not dominate $\text{c}$. We
assume that mechanisms outside the scope of this model ensure that users see only output which they are cleared to see, and that inputs are correctly classified.

This is expressed by saying that if two sequences of inputs are the same when viewed at a certain classification then the outputs will be the same when viewed at that classification.

\[
\forall \text{sys:SYSTEM} \bullet \text{sys} \in \text{secure} \iff \\
(\forall \text{class : Class}; \text{si1,si2 : seq IN}; \text{state:STATE} \\\n\mid (\text{si1,si2}) \in \text{same_ins class} \\\n\bullet (\text{second(sys (si1, state)), second(sys (si2, state)))} \in (\text{same_outs class}))
\]

The enunciation of the security requirement as a property of systems enables us to directly express the claim that (a mathematical model of) an implementation is secure. Some further elaboration of the type SYSTEM, probably making this type generic in various parameters, is likely to be necessary to ensure that this class of models is rich enough for the purposes in hand. The formal specifications of the design and implementation of the system will then be expressed as entities whose type is (some instance of) SYSTEM. The proposition that they are secure will then be expressible, and we hope provable, in Z, using the ICL Z proof tool.

2.4 SSQL specification

The proposed system is specified by the semantics of SSQL. To demonstrate that the SSQL specifications are secure in the sense defined above the SSQL specifications have to be formalised as definition of a particular SYSTEM.

This construction would be constructed from the formalised versions of the semantics for SSQL.

\[
\text{SSL-system : SYSTEM}
\]

\[
\text{true}
\]

2.5 correctness property

Formalisation of the correctness property depends upon the formalisation of the semantics of SSQL and hence upon a more detailed model of the interface to the system.

This can be achieved by making the critical requirement specification generic in those types the details of which are not relevant to the security requirements.

If more radical changes in the form of model used to express the full requirements are desirable then new base types will be defined and an interpretation map will be specified which shows how systems represented using the new base types can be interpreted as systems represented by the original types.

In this case we would suggest that the interpretation map should be reviewed formally (but not mathematically) to establish whether the map is a reasonable interpretation, and specifically whether it is liable to conceal insecurities, and used formally in the proofs that the “implementation” is secure.
correct : $\mathbb{P}$ SYSTEM

\[
correct = \{ \text{sys:SYSTEM} \\
\quad \forall \text{ si: seq IN} \\
\quad \bullet \text{ sys (si, sinit) = SSQL\_system (si, sinit)} 
\}\]

If the formal construction

### 3 A Formal Model of a Secure System Architecture

The system may be implemented minimising the critical function, by implementing a correct DBMS, which is not required to enforce security, and trusted FILTER, which will enforce the security constraints.

The DBMS is a subsystem which will provide direct access to the database. The FILTER mediates access to the database.

#### 3.1 Component Types

#### 3.1.1 The DBMS

The DBMS is modelled in a manner similar to the SYSTEM as a whole. In the target application we would not expect the query languages to be identical but for present purposes this will suffice. However, we would expect that the classifications attached to commands and data output will not be present on this interface (though classifications will appear in the data).

\[
\begin{align*}
\text{DBMS\_STATE} & \equiv \mathbb{P} \text{ Data} \\
\text{DBMS\_IN} & \equiv \text{Query} \\
\text{DBMS\_OUT} & \equiv \mathbb{P} \text{ Data} \\
\text{DBMS} & \equiv (\text{DBMS\_IN} \times \text{DBMS\_STATE}) \rightarrow (\text{DBMS\_STATE} \times \text{DBMS\_OUT})
\end{align*}
\]

#### 3.1.2 The FILTER

The filter takes queries in SSQL and responds to these queries by performing queries in SQL on the underlying database and sanitising the results.

It can therefore be modelled as a function which when supplied with an underlying DBMS returns a SYSTEM.

\[
\text{FILTER} \equiv \text{DBMS} \rightarrow \text{SYSTEM}
\]
3.2 Construction

Given the way the filter has been modelled, the construction of a system from a filter and a dbms is trivial.

\[
\text{construction} : \text{FILTER} \times \text{DBMS} \rightarrow \text{SYSTEM}
\]

\[
\forall \text{filter:FILTER; dbms:DBMS} \bullet \text{construction (filter, dbms)} = \text{filter dbms}
\]

3.3 Requirements on Components

3.3.1 Requirements on DBMS

Two properties are defined on the DBMS subsystem.
The first expresses the properties it must have in order for the system using it to be secure. It is not possible to establish the details of this property at this stage.

\[
\text{secure dbms} : \mathcal{P} \text{ DBMS}
\]

\[
\text{true}
\]

A stronger property is defined which is the property of complete correctness of the DBMS subsystem. It is reasonable to expect that this property will be principally based on the formal semantics of SQL.

\[
\text{correct dbms} : \mathcal{P} \text{ secure dbms}
\]

\[
\text{true}
\]

For maximum assurance the design of the secure database should exploit the underlying DBMS in a way which minimises the ways in which errors in the databaseis subsystem might make the overall system insecure. This will cause weakening of the property secure dbms and facilitate the construction of the necessary proofs.

3.3.2 Requirements on DBMS

Again two properties are relevant, security and correctness.

Defining the property secure filter is the activity which focusses on those aspects of the functionality of the filter which are critical to enforcing the security requirements on the resulting SYSTEM.

This contributes to the assurance of the system developed in two complementary but closely connected ways. Firstly it provides the information necessary to separate out those parts of the filter functionality which are security critical. Secondly it minimises the proofs relating to the filter as a whole necessary to establish security, and enables these proofs to be conducted either more rigorously, or more economically.
secure_filter : $P$ FILTER

true

The full correctness property for the filter will state that the filter will provide a system correctly implementing the semantics of SSQL provided that the DBMS correctly implements the semantics of SQL.

correct_filter : $P$ secure_filter

true

3.4 Architecture Correctness Conjectures

We are now in a position to formulate a conjecture expressing the claim that the architectural design modelled above suffices for the construction of secure systems.

Two conjectures are identified, one which states the security of the architecture and the other which states its correctness.

The first conjecture states that whenever a system is built using construction from a DBMS which is a secure dbms together with a FILTER which is a secure filter, then the resulting system will be secure.

\[
\forall \text{dbms:DBMS}; \text{filter:FILTER} \\
\text{dbms} \in \text{secure\_dbms} \land \text{filter} \in \text{secure\_filter} \\
\bullet \text{construction}(\text{filter}, \text{dbms}) \in \text{secure}
\]

The second conjecture expresses the claim that the architecture builds correct implementations.

\[
\forall \text{dbms:DBMS}; \text{filter:FILTER} \\
\text{dbms} \in \text{correct\_dbms} \land \text{filter} \in \text{correct\_filter} \\
\bullet \text{construction}(\text{filter}, \text{dbms}) \in \text{correct}
\]

4 A Model of a Secure Kernel

4.1 The Model

We may now proceed to the definition of a function which we believe to be a secure_kernel. We call this an implementation for present purposes. In another context we might regard this as a functional specification or a design.

The specification is sufficiently explicit that it could be manually translated to a similar program in a functional programming language, though it is more plausible as a rather abstract model of a kernel which would be elaborated somewhat before implementation.

The kernel adopts two measures to ensure that the application does not violate the security policy (the critical requirement). Firstly it ensures that the application does not have access to information
which the user is not cleared to see. This is modelled by the kernel supplying the application with a filtered copy of the classified data store from which highly classified data has been removed. Secondly it ensures that the application does not transfer information from highly classified data into data classified lower. This is modelled by the kernel filtering the classified data store returned from the application, discarding lowly classified data before using this to update the state of the system.

\[
\text{z} \quad \text{dbms\_implementation : } DBMS \\
\text{true}
\]

\[
\text{z} \quad \text{filter\_implementation : } FILTER \\
\text{true}
\]

5 System Correctness Proposition

Even though we have said nothing about the behaviour of the application, we have done enough formal modelling to establish that a system built from kernel\_implementation and an application using construction will be secure. If the system is defined as:

\[
\text{system : SYSTEM} \\
system = \text{construction(filter\_implementation, dbms\_implementation)}
\]

The claim that this system is secure may then be expressed:

\[
\text{z} \quad \vdash \ ? \quad system \in \text{secure}
\]

The proof of this conjecture is trivial given the two previous results.

6 Acknowledgements

Index of Formal Names

<table>
<thead>
<tr>
<th>Class</th>
<th>construction</th>
<th>correct</th>
<th>correct_dbms</th>
<th>correct_filter</th>
<th>Data</th>
<th>DBMS</th>
<th>dbms_implementation</th>
<th>DBMS_IN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>