Project: DRA FRONT END FILTER PROJECT

Title: Multi-level Formal Security Policy

Ref: DS/FMU/FEF/040 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
R. D. Arthan	WIN01		

Authorisation for Issue:

Name	*Function*	*Signature*	*Date*
R.B. Jones	HAT Manager		

Abstract: A formulation of a multi-level formal security policy for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0.2 Document Cross References

[1] DS/FMU/FEF/003. *Formal Security Policy.* G.M. Prout, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/039. *Proposal and Quotation for Phase 3.* R.D. Arthan, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/042. *Multi-level Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

[4] *Security Properties of the SWORD secure DBMS Design.* Simon Wiseman, DRA.

0.3 Changes History

Issue 1.1 (11 January 1994) First draft.

Issue *Revision* : *2.1* **(5 June 2016)** Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document is a draft of constitutes part of deliverable D17 of phase 3 of the FEF project, as described in [2].

1.2 Introduction

Phases 1 and 2 of the FEF project dealt exclusively with the formulation of the security policy defined in [1]. In that formulation SSQL queries are treated as containing information at a single security clearance. In phase 3, the intention is to generalise the policy to cater for queries which are structured objects containing components which may be at several different clearances. This document gives a formal specification of a behavioural model of systems and a formal specification of a security policy allowing for such multi-level queries.

In fact, this "security policy" is just the policy of [1] adapted so that the notion of two inputs or outputs "being the same at a given class" is no longer fixed, but is supplied as a parameter. This gives a simple framework in which some general reasoning about such policies can be carried out. The approach also concentrates attention on the "sameness" structure on the inputs and outputs (or input and output sequences). It is this structure which defines the meaning of assertions like "such-and-such a component of the data has such-and-such a sensitivity".

This document also contains the beginnings of a general analysis of information flows when a labelled component is extracted from an output sequence. This was first considered as a possible approach to the result-labelling property. However, further work on the specific details of SWORD (in [3]) suggests that more needs to be known about the actual intentions behind the component-labelling scheme than can be reflected easily in the general context of this document. The material might be relevant in investigating the constraints a trusted client of SWORD would have to obey when carrying out the sorts of down-grading mentioned in [4].

An index of the names used in the formal specification may be found in Section 5. A listing of the theory *fef040* created by processing this document using ProofPower may be found at the end of this document.

1.3 Setting Up

The following ProofPower instructions set up the new theory *fef040* and set the context for the proof tools. The parent theory is the theory *fef003* which contains the single level security policy and, in particular, the definition of the lattice of security classes.

SML

```
open_theory"fef003";
(force_delete_theory "fef040" handle _ => ());
new_theory "fef040";
new_parent "fef010";
push_pc "hol";
```

2 MULTI-LEVEL SECURITY POLICY

2.1 Behavioural Models of Systems

The "generic security policy" presented here would be instantiated as a property on behavioural models of systems. The inputs and outputs of systems will be thought of as structured objects whose components may be labelled with security classes. For example, an input might be an abstract syntax tree for an SSQL query with the nodes labelled with classes. An output might be an SSQL table with classes labelling the cells.

2.2 Classes

We use the lattice of security classes defined in [1] as the type *Class* together with operations *dominates*, *lub*, etc. We will need the notions of the down-set, $c\downarrow$, and the up-set, $c\uparrow$, of a class c. We write these operators with postfix syntax:

SML

```
declare_postfix(300, "↓");
declare_postfix(300, "↑");
```

HOL Constant

$$\$\downarrow : Class \to Class\ \mathbb{P}$$

$$\forall c \bullet \quad c\downarrow = \{\ d \mid c\ dominates\ d\ \}$$

HOL Constant

$$\$\uparrow : Class \to Class\ \mathbb{P}$$

$$\forall c \bullet \quad c\uparrow = \{\ d \mid d\ dominates\ c\ \}$$

We may later need the idea of the "orthogonal complement" of a set A of classes, which we shall write as $A^{\perp}$. $A^{\perp}$ is to be the set of classes which are are not comparable with any element of A.

SML

```
declare_postfix(300, "⊥");
```

HOL Constant

$$\$^{\perp} : Class\ \mathbb{P} \to Class\ \mathbb{P}$$

$$\forall A \bullet \quad A^{\perp} = \{\ c \mid \forall d \bullet d \in A \Rightarrow \neg\ c\ dominates\ d \wedge \neg\ d\ dominates\ c\ \}$$

2.3 Sameness Structures

In [1], the security policy is expressed in terms of relations *same_ins*, and *same_outs* parameterised by a security class. In [4], the non-interference property is expressed using a relation *identicalObjs* also parameterised by a class. [4] also uses a visibility predicate *visible* again parameterised by a class.

To generalise these ideas from [1, 4], it is convenient to define the notion of an equivalence relation, (an equivalence relation being one which is reflexive, symmetric and transitive).

HOL Constant

$$\textbf{Equivalence} : (\prime a \leftrightarrow \prime a)\ \mathbb{P}$$

$$Equivalence = Reflexive \cap Symmetric \cap Transitive$$

[Note: The above assumes that we are only concerned with equivalence relations on the entire type of objects. This may mean that if the input and output objects have to satisfy some invariant, then it has to be captured in a type definition.]

In [1, 4], two objects, x and y, are take to be identical viewed from a class, c, if they become the same when components whose classification is not dominated by c are "purged". I.e., x and y are taken as the same at c iff. they become identical when we treat components whose classification is not in $c\downarrow$ as being equal. To tackle the result labelling property, it is useful to have available the analogous notion for any set of classes, not just down-sets. To do this we introduce the idea of an *indexed equivalence relation*. This is to be a function, s associating with each set, A, of classes, an equivalence relation $s(A)$. We require that s be antimonotonic with respect to inclusion, i.e., the bigger the set A, the finer the equivalence relation $s(A)$:

HOL Constant

$$\textbf{IndexedEquiv} : (Class\ \mathbb{P} \rightarrow (\prime a \leftrightarrow \prime a))\ \mathbb{P}$$

$$\begin{array}{ll} IndexedEquiv = & \\ \{ & s \\ | & (\forall A \bullet\ s(A) \in Equivalence) \\ \land & (\forall A\ B \bullet\ A \subseteq B \Rightarrow s(B) \subseteq s(A))\} \end{array}$$

(Here s stands for "same" reflecting the terminology of [1]).

[Note: There are extra conditions one might choose to impose. E.g. that $s\{\}$ is the complete relation and that $s\ Universe$ is the identity relation. All the indexed equivalence one works with in practice satisfy $s(\bigcup U)=\bigcap(s(U))$, for arbitrary families of sets U. Thus $s(A)$ is determined by the values, $s\{c\}$, of s on singleton sets. However, the use of sets of classes seems to be technically convenient and allows one to consider various more or less pathological cases as well as well-behaved ones.]

Given a relation parameterised by individual classes we may lift it to an indexed relation. In the next section, this allows us to relate the multi-level security policy with the single level one of [1]. The lifting function is defined as follows:

HOL Constant

$$\textbf{LiftRel} : (Class \to ('a \leftrightarrow 'a)) \to (Class\ \mathbb{P} \to ('a \leftrightarrow 'a))$$

$$\forall R \bullet \quad LiftRel\ R = (\lambda A \bullet \bigcap \{\ r \mid \exists c \bullet\ c \in A \land r = R\ c\ \})$$

We will say that an indexed equivalence relation is *independent* if the following criterion holds:

HOL Constant

$$\textbf{Independent} : (Class\ \mathbb{P} \to ('a \leftrightarrow 'a))\ \mathbb{P}$$

$$Independent =$$
$$\{ \quad s$$
$$\mid \quad \forall A\ x \bullet (\exists y \bullet \neg(x,\ y) \in s(A)) \Rightarrow \exists z \bullet \neg(x,\ z) \in s(A) \land (x,\ z) \in s(A\ ^{\perp})\}$$

[Note: Above not used. Was intended to help with observations etc.]

2.4 Security Policy

The general multi-level security policy is parameterised by indexed equivalence relations on the inputs and outputs as defined in the previous section. It is convenient to have the policy as a property of arbitrary functions rather than just of functions mapping input lists to output lists.

HOL Constant

$$\textbf{x\_ml\_secure} : \quad (Class\ \mathbb{P} \to ('I \leftrightarrow 'I)) \to (Class\ \mathbb{P} \to ('O \leftrightarrow 'O)) \to ('I \to 'O)\ \mathbb{P}$$

$$\forall s_I\ s_O\ b$$
$$\bullet \quad b \in x\_ml\_secure\ s_I\ s_O$$
$$\Leftrightarrow \quad s_I \in IndexedEquiv$$
$$\land \quad s_O \in IndexedEquiv$$
$$\land \quad \forall c\ i_1\ i_2 \bullet$$
$$(i_1,\ i_2) \in s_I\ (c\downarrow)$$
$$\Rightarrow \quad (b\ i_1,\ b\ i_2) \in s_O\ (c\downarrow)$$

More commonly in practice we are interested in the special case where the indexed equivalences are derived by lifting a relation parameterised by individual classes. Thus we will more often use:

HOL Constant

$$\textbf{ml\_secure} : \quad (Class \to ('I \leftrightarrow 'I)) \to (Class \to ('O \leftrightarrow 'O)) \to ('I \to 'O)\ \mathbb{P}$$

$$\forall r_I\ r_O \bullet \quad ml\_secure\ r_I\ r_O = x\_ml\_secure\ (LiftRel\ r_I)\ (LiftRel\ r_O)$$

The following conjecture, which is fairly straightforward to prove once a few lemmas about down-sets etc. have been established, shows that the above multi-level security policy does generalise the single level one of [1].

```
SML
|val conj_040_1 = ⌜
|        ∀bm• bm ∈ secure ⇔ bm ∈ ml_secure same_ins same_outs
|⌝;
```

3 COMPONENT EXTRACTION

In this section certain ideas are explored which were intended originally to contribute to the formulation of the labelling property required for a multi-level secure relational database. It appears from initial investigation that a good relationship between these ideas and the flow policy is not easy to establish, and it turned out that a labelling property thought to be appropriate can be defined without recourse to these ideas. The section remains in the document *pro-tem*, pending consideration of whether this line can beneficially be further developed, but is not used as yet in subsequent documents.

The general way we are modelling structured multi-level objects requires us to use indirect means to handle the idea of a component of an object labelled with a class. We think of the component as being modelled by the function which maps an object to the class-value pair held in the component. Because the class component is itself a potential source of information flows it will often be more convenient to think of such a function as a pair of functions, one to returning the class and the other the value.

Given an arbitrary function on a set endowed with an indexed equivalence relation, we consider the classes at or above which information has flowed into the result of applying the function. This notion may be formalised as follows:

HOL Constant

$$\textbf{Influenced}: (\mathit{Class}\ \mathbb{P} \to ('a \leftrightarrow 'a)) \to 'a \to ('a \to 'b) \to \mathit{Class}\ \mathbb{P}$$

$$\forall s\ x\ f \bullet$$
$$\mathit{Influenced}\ s\ x\ f =$$
$$\{\ c \mid \exists y \bullet (x,\ y) \in s(\sim(c\uparrow)) \wedge \neg f(x) = f(y)\ \}$$

(Here $\sim A$ is ProofPower-HOL notation for the complement of the set A.)

That is to say $\mathit{Influenced}\ s\ x\ f$ is the set of classes c for which there is some state y, such that f distinguishes x and y, but x and y appear to be identical when viewed from any class which does not dominate c.

We model the process of examining a labelled component of an object in the output of a system by two functions, C and V say: C computing the class; and V computing the value. The information-flow properties of such a pair will be considered relative to the clearance, c say of the client to whom the result of the observation may be revealed. We will call such a triple comprising c, C and V an *observation*:

SML

```
declare_type_abbrev("OBSERVATION",["'a", "'b"],
        ⌜: Class × ('a → Class) × ('a → 'b)⌝);
```

The labelled value produced when an object is observed is defined by the following function (which converts the pair of functions into a function returning pairs):

HOL Constant

$$\textbf{ObservedValue} : (\prime a, \prime b)\ OBSERVATION \to \prime a \to (Class \times \prime b)$$

$$\forall c\ C\ V\ x \bullet$$
$$ObservedValue\ (c,\ C,\ V)\ x = (C\ x,\ V\ x)$$

We can think of the labelled values resulting from observations as (single-level) classified objects using the following parameterised equivalence relation, (cf. the filtering operation which are performed on database cells in the single-level formulation of SWORD).

HOL Constant

$$\textbf{SameLabVal} : Class \to (Class \times \prime b) \leftrightarrow (Class \times \prime b)$$

$$\begin{array}{lll}\forall c \bullet & SameLabVal\ c = & \\ & \{ & ((c_1,\ v_1),\ (c_2,\ v_2)) \\ & | & (c_1 = c_2) \\ & \wedge & (c\ dominates\ c_1 \Rightarrow v_1 = v_2)\ \}\end{array}$$

We can now consider the question of when the process of making an observation is secure. When a client makes an observation, we assume that some trusted part of the overall system will check the class which is returned and will withhold the value from clients who are not cleared to see it. Thus the value may depend on information a client is not cleared to see, but this does not matter provided the class prevents the client from seeing the value. However, since the class is itself intended to be part of what the client making the observation sees, the class must not depend on information the client is not allowed to know. We will call an observation *bounded* if the following holds of it:

HOL Constant

$$\textbf{BoundedObs} : (Class\ \mathbb{P} \to (\prime a \leftrightarrow \prime a)) \to (\prime a, \prime b)\ OBSERVATION\ \mathbb{P}$$

$$\begin{array}{lll}\forall s \bullet & BoundedObs\ s = & \\ & \{ & (c,\ C,\ V) : (\prime a, \prime b)\ OBSERVATION \\ & | & \forall x \bullet \quad Influenced\ s\ x\ C \subseteq c\downarrow \\ & & \wedge \quad Influenced\ s\ x\ V \subseteq (C\ x)\downarrow\ \}\end{array}$$

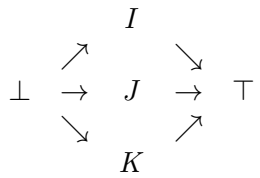
That boundedness is necessary for an observation to be secure in a certain sense, is as asserted in the following conjecture:

SML

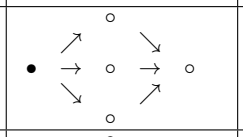
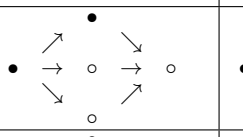
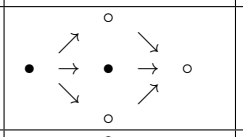
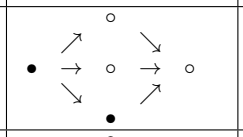
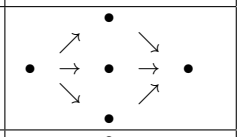
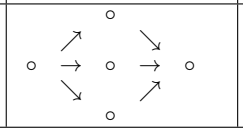
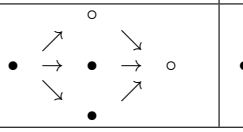
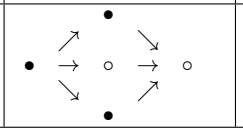
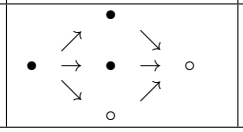
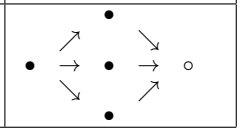
$$\begin{array}{l}val\ \textbf{conj\_040\_2} = \ulcorner \\ \forall s \bullet \quad s \in IndexedEquiv \\ \Rightarrow \quad \forall c\ C\ V \bullet \quad ObservedValue\ (c,\ C,\ V) \in x\_ml\_secure\ s\ (LiftRel\ SameLabVal) \\ \qquad\qquad \Rightarrow \quad (c,\ C,\ V) \in BoundedObs\ s \\ \urcorner;\end{array}$$

However, boundedness is not sufficient in general. The problems arise with dependencies between the data which is visible at incomparable classes (e.g., arising from a state invariant linking values

stored under incomparable classes). E.g. assume the lattice of classes comprises $\bot$, $\top$ and three incomparable classes I, J, and K, as shown in the following diagram:



As c ranges over the possible classes, the sets $c{\downarrow}$ which appear in the definition of *ml_secure* and the sets $\sim(c{\uparrow})$ which appear in the definition of *Influenced* are as shown in the following table:

c	$\bot$	I	J	K	$\top$
$c{\downarrow}$					
$\sim(c{\uparrow})$					

Now, let us assume that objects are triples of integers, (x_I, x_J, x_K), and that the indexed equivalence, s, on these objects is obtained by classifying each component x_c at c (so that clients at class $\bot$ can see nothing, clients at classes I, J and K can see x_I, x_J and x_K, respectively, and clients at class $\top$ can see everything). Let us assume in addition that the objects are constrained to satisfy the invariant $x_I + x_J + x_K = 0$. Consider any function V with domain the set of triples satisfying this invariant. Because of the invariant, the value of V on any object x is determined by the values of any two of the three components of x. By reference to the table showing the values of $\sim(c{\uparrow})$ above, we find that for any state x, *Influenced* $s\ x\ V = \{\bot\}$. However, it is clearly not secure to classify an arbitrary function on the set of objects as computing a value at class $\bot$. E.g. the function V_I which maps (x_I, x_J, x_K) to x_I must be classified at or above I in order to be secure.

4 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

```
SML
pop_pc();
```

5 INDEX