A Security Model of Dynamic Labeling Providing a Tiered Approach to Verification*†

Simon N. Foley
Department of Computer Science
University College
Cork, Ireland
s.foley@cs.ucc.ie

Li Gong and Xiaolei Qian
Computer Science Laboratory
SRI International
Menlo Park, California 94025, USA
{gong,qian}@csl.sri.com

Abstract

In the proposed mandatory access control model, arbitrary label changing policies can be expressed. The relatively simple model can capture a wide variety of security policies, including high-water marks, downgrading, separation of duties, and Chinese Walls. The model forms the basis for a tiered approach to the formal development of secure systems, whereby security verification can be spread across what makes up the reference monitor and the security requirement specification. The advantage of this approach is that once a trusted computing base (TCB) is in place, reconfiguring it for different security requirements requires verification of just the new requirements. We illustrate the approach with a number of examples, including one policy that permits high-level subjects to make relabeling requests on low-level objects; the policy is multilevel secure.

1 Introduction

Information-flow policy models that support dynamic labeling, where information labels can change in time, have been shown to capture a greater variety of security requirements than can models where labeling is static. One of the earliest examples of a system supporting dynamic labeling is ADEPT-50 [17], where an object's label can rise to reflect the classification of the data written to it by a subject. It is well known that label changes can often be exploited for covert channels. For example, the effect of a label change to a low-level object, requested by a high-level subject, may be visible to a low-level subject in the form of a (necessary) change of access permissions to reflect its new label.

*This work was supported in part by the U.S. Department of Defense Advanced Research Projects Agency and the U.S. Air Force Rome Laboratory under contract F30602-94-C-0198, and in part by the U.S. National Science Foundation under grant ECS-94-22688.

†An abridged version of this report is due to appear in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.

Covert channels in dynamic labeling schemes can be minimized or avoided entirely if the nature of the label change is constrained. A widely used strategy is to permit only *upgrades from below*. For example, in [18, 15], requests to upgrade the label of an object may be done only if the classification of the subject is *dominated* by the current classification of the object; label changes may be based only on information that may already flow to the label classification. Further examples of constraining relabeling include the Compartmented Mode Workstation [19] and SERCUS [10], a secure document-handling system where drafts and private documents can have dynamic labels, but once made public they become static. Other work has shown how constrained dynamic labeling policies can be used to capture specific security requirements. Examples are Chinese Wall policies [6, 12, 14] and segregation of duties [5]. Implicit in this work is some form of high water mark mechanism, whereby a subject's label classification rises to access requested information, thereby excluding access to other information. The reader is referred to [13] for a general introduction to, and discussion on, dynamic upgrading policies.

We propose a general security model for dynamic label changing. The relatively simple model is of Bell-LaPadula style and can capture all the policies mentioned above. Of note, is that details about how labels may be changed is considered part of the information flow and security policy (along with the usual lattice of classifications). The security model may be thought of as being 'parameterized' by this policy: defining what it means for a system to be secure according to lattice and relabel rules. This aspect of the model is different from many existing dynamic labeling approaches; in these cases specific rules on how labels may change are typically encoded as part of the system model. Existing models that can be interpreted as having the effect of supporting parameterized relabeling rules include McLean's MAC model [11] and the expressive labels proposed by Gong and Qian [8]. With the former, subjects may be given the right to arbitrarily modify particular object labels. In [8] information about future label changes may be encoded into an object's label by subjects.

It is this grouping together of lattice and relabel rules into an abstract security policy that provides the basis for our tiered approach to security verification. If we build a security kernel that can support a subset of all possible lattice and relabel rules, then security verification can be performed in two stages. The first stage is to verify that the low-level kernel implementation is secure against the axioms of the model, and the second is verification that a particular policy is supported by the kernel. Reconfiguring the kernel for a new policy requires verification of just the new policy.

Section 2 defines what we mean by a dynamic relabeling policy and proposes a MAC security model based on these policies. The model has easily recognizable Bell-LaPadula origins; this gives us confidence that it is a reasonable characterization and that restricted versions could be implemented in practice. Section 3 outlines how the model might be interpreted and verified in practice. While our relabel policies have the expressiveness of finite state automatons, their potential is illustrated in Section 4, by the development of two security policies. The first relabeling policy supports upgrade paths, where a low-level object can be marked so that when deleted by a low-level subject, it is upgraded to high, rather than actually deleted. The policy is developed at an abstract level, rather than in the low-level detail of the original implementation in [2]. The second policy is a multilevel Chinese Wall policy applied to users who are members of different groups. Both these policies can be verified as upgrade-from-below policies and thus can be enforced by any security kernel verified to support generic upgrade from-below-policies.

We believe that it is (relatively) straightforward to build a secure kernel supporting upgrade-from-below policies. Sections 5 considers the problems of supporting more general upgrade policies, where the requests for upgrade come from above. Adopting our tiered verification approach, Section 5 defines verification conditions that the policy should have so that it can be enforced by a general security kernel. The essence of these conditions on the abstract policy is that it is not possible for changes in low-level labels (requested by high-level subjects) to be visible to low-level subjects.

2 A Model for Relabeling

2.1 Relabeling Policies

Given a lattice, with security classes L and partial order $\leq$, a relabeling policy should define the circumstances under which information at one class may be relabeled to another. A *relabeling function* is a partial function $f : relabel$ ($relabel \mathrel{\widehat{=}} L \nrightarrow (L \nrightarrow L)$), with an interpretation that $f(s)$ defines the possible relabeling when requested (by a subject) at class $s \in \operatorname{dom} f$. Evaluation $f(s)(a) = b$ means that when requested at class s, information at class a may be relabeled as class b. A *relabeling policy* is defined as a set R of such functions.

Example 1 A simple relabeling policy for the lattice with ordering $\mathtt{lo} < \mathtt{hi}$ provides partial relabeling functions

$$\begin{array}{l}\mathtt{upgrade} \mathrel{\widehat{=}} \lambda(s : L) \mid s = \mathtt{lo} \bullet (\lambda(a : L) \mid a = \mathtt{lo} \bullet \mathtt{hi}) \\ \mathtt{downgrade} \mathrel{\widehat{=}} \lambda(s : L) \mid s = \mathtt{hi} \bullet (\lambda(a : L) \mid a = \mathtt{hi} \bullet \mathtt{lo})\end{array}$$

The syntax for lambda abstraction ($\lambda\, s : L \mid P(s) \bullet E(s)$) specifies that the function is defined only for values of s that satisfy property $P(s)$. With this policy, upgrades may be done only from `lo`: a typical implementation strategy for avoiding covert channels. Note, however, that we will not insist in our general model that upgrades must always be done in this way: in certain cases an upgrade from above may be considered an appropriate relabel policy.

A slightly more sophisticated relabeling policy might require that high-level information to be downgraded be first relabeled for inspection (by a security officer) and then, if appropriate, downgraded by the officer. The lattice can be defined by the power-set lattice of $\{h, d\}$, where $\{\}$ represents low, $\{h\}$ represents high, and $\{d\}$ represents the label of information to be downgraded. The relabeling policy is defined by functions

$$\begin{array}{l}\mathtt{submit} \mathrel{\widehat{=}} \lambda(s : \mathbb{P}\, L) \mid s = \{h\} \bullet (\lambda(a : \mathbb{P}\, L) \mid a = \{h\} \bullet \{d\}) \\ \mathtt{downgrade} \mathrel{\widehat{=}} \lambda(s : \mathbb{P}\, L) \mid s = \{d\} \bullet (\lambda(a : \mathbb{P}\, L) \mid a = \{d\} \bullet \{\})\end{array}$$

△

Relabeling policies have the expressiveness of finite-state automatons, where security classes correspond to states, and relabel functions to state transitions. Thus, it is not surprising that a wide variety of security policies can be captured using relabeling, as will be illustrated in Section 4.

2.2 System Model

A system is an abstract state machine, described in terms of a finite set of subjects S, a finite set of objects O, a finite lattice of security classifications $(L, \leq)$, a relabeling policy R, and a set A of access rights that subjects can have on objects (for our purposes, the rights *read* and *write*).

A (security) state v of the system is given by the tuple $(M, amin, vmax, lab)$, where $M : S \times O \to \mathbb{P}\, A$ is an access matrix, indicating the rights that subjects currently have on objects, $lab : O \to L$ gives the security classification associated with each object, each subject $s \in S$ is associated with an interval from the lattice, $vmax(s)$ (view maximum) gives the highest class of information that the subject is trusted to read, and $amin(s)$ (alter minimum) gives the lowest class of information the subject is permitted to write, that is, subjects may have partial trust in the sense of [4]. We have for every subject $s : S$ that $amin(s) \leq vmax(s)$. Let V denote the set of all states.

A subject can request that the system moves from one state to another, under a transition operation op. The set of all transition operations that can change the security state is given by the set T. Transitions include operations to grant accesses, create objects, and so forth. However, for the purposes of this paper, we are interested only in transitions that result in relabeling of information (subject or object labels). We call these transition operations *primitive relabel operations*, and a function $\mathcal{R} : T \to R$ associates a relabeling function $\mathcal{R}(op)$ from relabel policy R with each transition operation $op \in T$.

Thus, $(\mathcal{R}(op)\; s) = f$ defines that operation op, when requested at class s, may relabel information according to function $f(s)$. Which objects or subjects actually get relabeled is part of the implementation. The behavior of the state transition functions is defined by $\Sigma : S \times T \times V \to V$, where a subject $s : S$ requesting operation $op : T$ in state v moves the system to state $v' = \Sigma(s, op, v)$.

2.3 Security Axioms

Axiom 1 A state is secure if the accesses currently held by subjects on objects do not violate the lattice policy: for all states v reachable from the initial state, then

$$\begin{array}{l} \forall\, s : S;\ o : O \bullet \\ \qquad read \in M[s,o] \Rightarrow lab(o) \leq vmax(s) \;\wedge \\ \qquad write \in M[s,o] \Rightarrow amin(s) \leq lab(o) \end{array}$$

$\diamond$

This axiom is simply the partial trust adaptation of the star property and simple security condition. As with the Bell-LaPadula model [1], we implicitly assume that an object does not change from state to state unless someone has write access to it.

A transition operation is secure if any changes made to the security state conform to the relabeling policy. Specifically, for each transition $\Sigma(s, op, v) = v'$, where $v = (M, amin, vmax, lab)$ and $v' = (M', amin', vmax', lab')$, then for every subject x and object o:

- **Axiom 2** If $lab(o) \neq lab'(o)$, then the requesting subject s must be trusted to make this alter request, that is,

  $$lab'(o) = op(amin(s))\, lab(o)$$

  Note that we relax our syntax, using $op(i)$ as a shorthand for $(\mathcal{R}(op))(i)$, where no ambiguity can arise. $\diamond$

- **Axiom 3** If $amin(x) \neq amin'(x)$, then the requesting subject s must be trusted to make this alter request, that is,

  $$amin'(x) = op(amin(s))\, amin(x)$$

  And similarly, if $vmax(x) \neq vmax'(x)$ then

  $$vmax'(x) = op(amin(s))\, vmax(x)$$

  $\diamond$

Example 2 General upgrading (for objects) could be provided for by a transition operation $\texttt{UpgradeO}(o, b)$, which corresponds to a request that object o be upgraded from its current class to class b. The relabeling policy for this operation is described in terms of a family of relabel functions of the form $\texttt{upgrade}(b)$, where

$$\texttt{upgrade}(b) \mathrel{\widehat{=}} \lambda(s : L) \bullet (\lambda(a : L) \mid s \leq a < b \bullet b)$$

and we have $\mathcal{R}(\texttt{UpgradeO}(o, b)) = \texttt{upgrade}(b)$. In this case, upgrade is not defined if it is not requested from below. A similar relabeling function could be devised for an upgrade transition on a particular subject's *amin* and *vmax*. $\triangle$

Example 3 Low-level information may be marked, so that when deleted it is upgraded to a higher level rather than actually deleted. Consider a lattice with classes `lo` and `hi`, and a special class `mlo`, which represents `lo` information that has been marked for upgrade. The lattice has ordering $\texttt{lo} \leq \texttt{mlo} \leq \texttt{hi}$, and the relabeling operations can be defined in terms of the upgrade function from Example 2,

$$\texttt{mark} \mathrel{\widehat{=}} \texttt{upgrade}(\texttt{mlo})$$
$$\texttt{mdel} \mathrel{\widehat{=}} \lambda(s : L) \mid s \leq \texttt{mlo} \bullet (\lambda(a : L) \mid a = \texttt{mlo} \bullet \texttt{hi}))$$

Note that with this interpretation, a typical low-level subject s should run with partial trust $amin(s) = \texttt{lo}$, $vmax(s) = \texttt{mlo}$, reflecting that the subject is trusted to simultaneously handle all kinds of low-level information (both marked and unmarked). In Section 4 we will illustrate how a more general marking policy can be specified. $\triangle$

Our axioms give *necessary* but not sufficient conditions for security. We view the model as one that provides a guideline on how to approach implementing general relabeling policies. As with

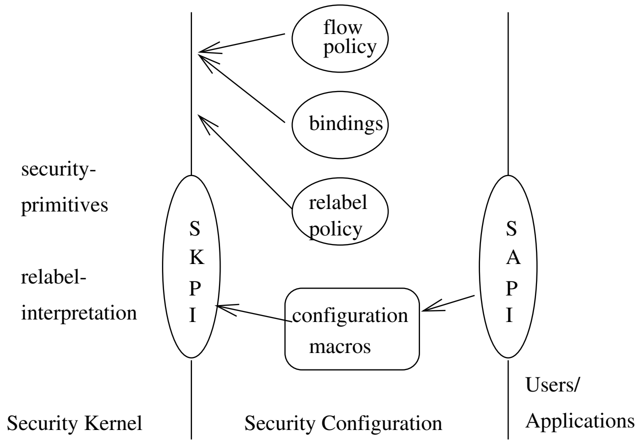

Figure 1: Security Architecture

any access-control-based model, covert channel analysis of the implementation will be necessary. For example, a relabeling must leave the system in a secure state, and this may result in subjects loosing access rights to reflect the relabeling. Any covert channel analysis will depend on what flows are considered acceptable or unacceptable. If an upgrading function permits low-level upgrades requested from high-levels then it is possible that a covert channel will result from high to low. One could argue that such a flow has been implicitly specified in the policy and should be acceptable (in the same way that a trusted downgrade is acceptable). If it is not, then either the implementation is analyzed to ensure that these covert channels do not exist, or the implementation will accept only a restricted class of policies. A simple example of this will be considered in Section 5. Some work already exists on noninterference models for particular dynamic labeling policies, including downgrading [16] and Chinese Walls [6]. Formulating general relabel policies in terms of noninterference is a topic for future research.

3 Interpreting the Security Model

In describing how the proposed security model might be interpreted in practice, our aim is to show that it should be possible to build and certify a security architecture that can support a wide range of security policies. The basic security architecture is outlined in Figure 1.

In our model we have focused only on primitive relabeling operations, but a typical security kernel model also provides operations for creation, granting access rights, and so forth. In general, an implementation of our security model should provide some form of relabel interpretation function, which when invoked from outside the kernel, appropriately interprets a requested primitive relabel operation passed as a parameter (performing relabeling on appropriate objects). Some consistent interpretation will have to be made on what objects and subjects the relabel should be applied to. In Example 2, operation `UpgradeO`(o, b) is expected to modify only object o. The primitive kernel operations make up the security kernel programming interface (SKPI), on top of which the (untrusted) operating system and applications are built.

The security policy for a particular enterprise is configured as a lattice, appropriate bindings for users, a relabel policy, and a number of configuration macros. These macros construct a security application programming interface (SAPI): a set of operations through which users and applications can have access to the security policy. These macros map abstract security operations down onto primitive operations. Given the simple policy in Example 3, an abstract delete operation that either upgrades (if marked) or actually removes the object might be described as

```
sapi_Delete(s)(o):=
  if pre (mdel amin(s) lab(o)) then
    RelabelO(o,mdel)
  else
    DeleteO(o)
```

The macro runs at the class s of the requester. The precondition `pre (mdel s lab(o))` tests whether a relabel function is defined for this particular request class and label. In this example, we assume that the kernel provides a general object relabeling primitive, where `RelabelO(o,f)` represents a transition operation, with $\mathcal{R}(\texttt{RelabelO}(\texttt{o},\texttt{f})) = \texttt{f}$, and relabels only object `o` according to function `f`. Transition operation `DeleteO(o)` is also provided by the kernel.

A single invocation of a configuration macro may make label changes to a number of different objects and subjects. It is possible that intermediate states result in bindings that are considered nonsecure or inconsistent. We therefore regard macros as atomic.

If the security kernel is secure, then these configuration macros can be viewed as relatively untrusted, in the sense of preserving the flow policy. They provide an abstraction of the kernel primitives for applications and users. However, one must assume that the policy they describe is *credible* and is the policy that is intended: a configuration that specifies that `hi` information may be arbitrarily downgraded is a valid policy, but it does permit flows that are surprising. Thus, relabeling policies may require additional analysis to determine that they are sensible. For example, if relabeling functions generate downward flows, they should be carefully used.

Our goal is to design kernels that support restricted classes of security policy, and require minimal or no security analysis of the configuration macros. In the case of a secure kernel that supports only upgrade-from-below policies, then credible configuration macros can be untrusted, and the security policy is enforced. The resulting system is secure since any macro will run at the level of its requester and, regardless of its behavior, the security kernel will permit upgrades only from below. In Section 5 a more general upgrade-only architecture will be described that permits certain policies supporting upgrades from above, allows untrusted but credible configuration macros, and is secure.

4 Specifying Relabel/Flow Policies

A general mark-for-upgrade policy and a complex Chinese Wall policy illustrate how complex relabel policies can be constructed within our model.

4.1 Mark-for-Update Policies

In Example 3 we described a simple upgrade policy, where marked low-level objects are upgraded to a high-level, when deleted by a low-level subject. A mark-for-upgrade scheme, defined here, that will work for any base policy, and provides upgrade paths similar to those provided by the $\sqcup$ operator in [8].

Our approach to policy specification is to represent all kinds, or classes, of information in the system as a datatype. It is in terms of this datatype that the flow relation and relabeling policies are defined. The policy is concerned with two kinds of information: simple (unmarked) classes and structured (marked) classes indicating an upgrade path. Given a base lattice of sensitivity levels $(L, \leq)$, the security classes for the mark-for-upgrade policy are defined using datatype

$$\mathit{Class} ::= \mathtt{C}\langle\!\langle L \rangle\!\rangle \mid \mathtt{M}\langle\!\langle L \times (\mathit{Class}) \rangle\!\rangle$$

A label of class $(\mathtt{C}\ a)$ represents simple information with sensitivity level a. A label $(\mathtt{M}\ a\ x)$ represents marked information with an upgrade level a; the actual level of the information is given in x. For example, given the typical multilevel secure (MLS) lattice of levels $\mathtt{unclass} < \mathtt{secret} < \mathtt{topsecret}$, an object label $(\mathtt{M}\,\mathtt{topsecret}\,(\mathtt{C}\,\mathtt{unclass}))$ represents information at class $\mathtt{unclass}$, with an upgrade path of just $\mathtt{topsecret}$.

Upgrade paths are strictly increasing in sensitivity. Given an arbitrary class x, define its effective class $\mathit{eff}(x)$, and highest upgrade class $\mathit{top}(x)$, as

$$\begin{array}{rl} \mathit{top}(\mathtt{C}\ a) \mathrel{\widehat=} a & \quad \mathit{top}(\mathtt{M}\ a\ x) \mathrel{\widehat=} a \\ \mathit{eff}(\mathtt{C}\ a) \mathrel{\widehat=} a & \quad \mathit{eff}(\mathtt{M}\ a\ x) \mathrel{\widehat=} \mathit{eff}(x) \\ \mathit{marked}(\mathtt{C}\ a) \mathrel{\widehat=} \mathit{False} & \quad \mathit{marked}(\mathtt{M}\ a\ x) \mathrel{\widehat=} \mathit{True} \end{array}$$

Thus, a particular class $x \in \mathit{Class}$ is a legal label if $\mathit{vldlbl}(x)$, where

$$\begin{array}{rcl} \mathit{vldlbl}(\mathtt{C}\ a) & \widehat= & \mathit{True} \\ \mathit{vldlbl}(\mathtt{M}\ a\ x) & \widehat= & a > \mathit{top}(x) \wedge \mathit{vldlbl}(x) \end{array}$$

The set of all labels is $\mathit{labels} \mathrel{\widehat=} \{x : \mathit{Class} \mid \mathit{vldlbl}(x)\}$. Base lattice $(L, \leq)$ is finite, and thus set labels is finite, since upgrade paths are strictly increasing. Other useful functions on the datatype Class include inserting a class (if possible) into an upgrade path and upgrading the effective class (if possible).

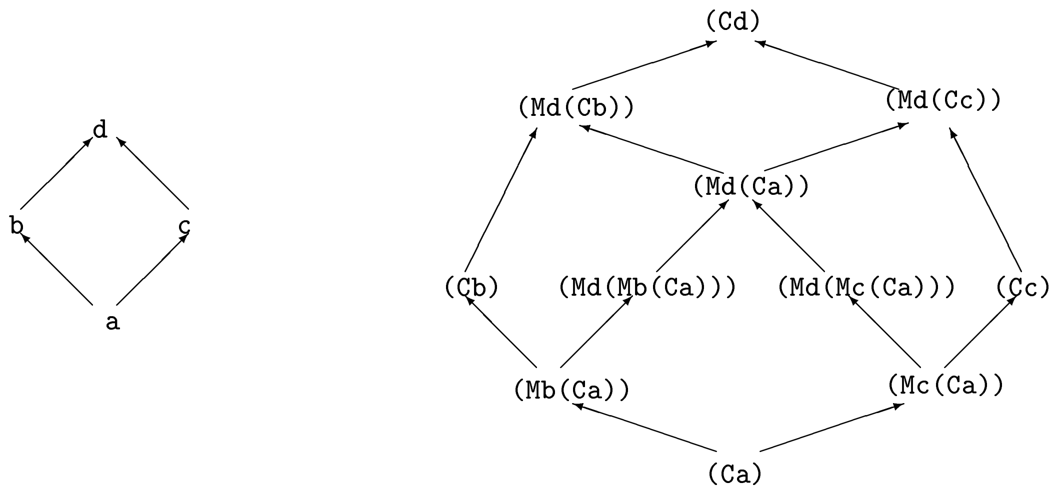

Figure 2: Base Lattice and Resulting Upgrading Lattice

$$
\begin{aligned}
insert &: \quad Class \to L \to Class \\
insert(\mathtt{C}\,a)\ b &\ \widehat{=}\ \ \text{if}\ a < b\ \text{then}\ (\mathtt{M}\,b\ (\mathtt{C}\,a)) \\
&\qquad \text{else}\ (\mathtt{C}\,a) \\
insert(\mathtt{M}\,a\ x)\ b &\ \widehat{=}\ \ \text{if}\ a < b\ \text{then}\ (\mathtt{M}\,b\ (\mathtt{M}\,a\ x)) \\
&\qquad \text{else}\ (\mathtt{M}\,a\ (insert\ x\ b))
\end{aligned}
$$

$$
\begin{aligned}
up &: \quad Class \to Class \\
up(\mathtt{M}\,a\ (\mathtt{C}\,b)) &\ \widehat{=}\ \ (\mathtt{C}\,a) \\
up(\mathtt{M}\,a\ (\mathtt{M}\,b\ x)) &\ \widehat{=}\ \ (\mathtt{M}\,a\ (up\ (\mathtt{M}\,b\ x)))
\end{aligned}
$$

An ordering relation $\leq_m$ can be constructed between these classes, based on the ordering of $(L, \leq)$.

$$
\begin{aligned}
(\mathtt{C}\,a) \leq_m (\mathtt{C}\,b) &\ \widehat{=}\ \ a \leq b \\
(\mathtt{C}\,a) \leq_m (\mathtt{M}\,b\ y) &\ \widehat{=}\ \ (\mathtt{C}\,a) \leq_m y \\
(\mathtt{M}\,a\ x) \leq_m (\mathtt{C}\,b) &\ \widehat{=}\ \ a \leq b \\
(\mathtt{M}\,a\ x) \leq_m (\mathtt{M}\,b\ y) &\ \widehat{=}\ \ a \leq b\ \wedge \\
&\qquad \text{if}\ top(x) < b\ \text{then}\ x \leq_m (\mathtt{M}\,b\ y) \\
&\qquad \text{elseif}\ top(y) < a\ \text{then}\ (\mathtt{M}\,a\ x) \leq_m y \\
&\qquad \text{else}\ x \leq_m y
\end{aligned}
$$

If $(L, \leq)$ is a lattice, we can prove that $(labels, \leq_m)$ is a lattice. Figure 2 gives an example of an instantiated lattice. There are two families of relabel functions.

$\texttt{mark} : L \nrightarrow \mathit{labels} \nrightarrow \mathit{labels} \nrightarrow \mathit{labels}$
$\texttt{mdel} : \mathit{labels} \nrightarrow \mathit{labels} \nrightarrow \mathit{labels}$

Relabel function $\texttt{mark}(u)$ defines the relabeling required when a sensitivity level u is to be added to the upgrade path of an object.

$$\texttt{mark}(u) \mathrel{\widehat{=}} \lambda(s : \mathit{labels}) \bullet \lambda(o : \mathit{labels}) \mid s \leq_m o \bullet \mathit{insert}(o, u)$$

Note that a relabeling is defined only when the subject's class is dominated by the object's class, that is, an upgrade from below. Function `mdel` defines the relabeling (upgrading) required for marked objects. Such upgrades are possible only from below.

$$\texttt{mdel} \mathrel{\widehat{=}} \lambda\, s : \mathit{labels} \bullet (\lambda(x : \mathit{labels}) \mid \mathit{marked}(x) \wedge s \leq_m x \bullet \mathit{up}(x))$$

The configuration macros are straightforward to define for this policy. Application interface call `sapi_Mark(o,u)` marks the object `o` to include `u` in its path. The call `sapi_Delete(s)(o)` deletes (by either upgrading or actually deleting) object `o`.

```
sapi_Mark(s)(o,u):=
  if pre (mark(u) amin(s) lab(o)) then
    RelabelO(o,mark(u))

sapi_Delete(s)(o):=
  if pre (mdel amin(s) lab(o)) then
    RelabelO(o,mdel)
  else
    DeleteO(o)
```

An appropriate strategy must be devised for determining suitable trust for individual subjects. For example, a subject with unclassified clearance is trusted to handle (read and write) any information with an effective class of `unclass`, and thus should have (`C unclass`) as *amin* and *vmax* as (`M topsecret (C unclass)`). For the policy in Figure 2, binding a subject with effective class `c` to interval ((`C c`), (`M d (C c)`)) ensures that it can read all unclassified information, even when it is marked to be upgraded to the disjoint class `b`. This aspect of the security policy (determination of appropriate bindings for subjects) must also be specified as part of the configuration macros.

4.2 General Chinese Wall Policies

A variety of approaches have been proposed for implementing Chinese Wall policies using multilevel techniques [12, 6, 14]. Implicit in these approaches is some form of high-water mark mechanism, whereby a subject's clearance rises to access organization information, if there is no conflict of interest. Here, we specify a very general relabeling policy that combines Chinese Wall, sensitivity levels and group membership.

First, we assume that there exist sets ORG, GRP, and L. The set $ORGS$ represents the set of all possible organizations. Conflicts between organizations are specified by the relation $\wr$, where $a \wr b$ means that organizations a and b are in competition with one another. From this we can define the set of all possible, sets of nonconflicting organizations as

$$ORGNC \mathrel{\widehat{=}} \{orgs : \mathbb{P}\, ORG \mid (\forall\, o, o' : orgs \bullet \neg\, (o \wr o'))\}$$

The set GRP represents the set of all possible groups of which subjects may be members. Consultants work for different organizations, but we must ensure no conflict of interest across the consultants of any given group. Furthermore, the multilevel ordering $(L, \leq)$ is expected to be enforced.

Our approach is that when a consultant (subject) wants to access a given organization, he must request an upgrade to an appropriate level. If the relabeling is permitted, then the new access must also be considered in all subsequent relabeling requests made by other subjects in the same group. We solve this problem by having a special environment object for each group, with a label that reflects the current accesses of subjects in that group. To consult for an organization, the consultant subject first requests that the label for his group object be updated, and then updates his label from the group label. This is specified in detail below.

We identify two kinds of information in this policy: information about groups and general information.

$$\begin{array}{lll} \mathit{Class} & ::= & \mathtt{C}\langle\!\langle L \times GRP \times (\mathbb{P}\, GRP) \times ORGNC \rangle\!\rangle \\ & \mid & \mathtt{G}\langle\!\langle GRP \times ORGNC \rangle\!\rangle \end{array}$$

To keep the policy simple, we do not distinguish between organization and consultant information. Class $(\mathtt{C}\, l\, g\, gs\, os)$ represents general information at sensitivity level l that was created by a subject in group g, and contains information sourced from groups gs and organizations os. A typical label binding for an organization o dataset would initially be $(\mathtt{C}\, l\, g\, \{\}\, \{o\})$: of course, datasets composed of aggregate organization information are permitted, so long as there is no conflict of interest in their resulting label. Typical bindings for a subject would be $amin = (\mathtt{C}\, l\, g\, \{\}\, os)$, where g is the group of which the subject is a member and os the organizations for which the subject is currently consulting. Having $vmax = (\mathtt{C}\, l\, g\, gs\, os)$ permits a subject to communicate with other subjects in different groups, but only if no conflict of interest results. Class $(\mathtt{G}\, g\, os)$ represents information about group g: that its subjects have accessed organizations os. The flow ordering between these classes is defined as

$$\begin{array}{rcl} (\mathtt{C}\, l\, g\, gs\, os) \leq_c (\mathtt{C}\, l'\, g'\, gs'\, os') & \widehat{=} & (l \leq l') \wedge (g = g') \wedge (gs \subseteq gs') \wedge (os \subseteq os') \\ (\mathtt{C}\, l\, g\, gs\, os) \leq_c (\mathtt{G}\, g'\, os') & \widehat{=} & \mathit{False} \\ (\mathtt{G}\, g\, os) \leq_c (\mathtt{G}\, g\, os) & \widehat{=} & (g = g') \wedge (os \subseteq os') \\ (\mathtt{G}\, g\, os) \leq_c (\mathtt{C}\, l'\, g'\, gs'\, os') & \widehat{=} & (g = g') \wedge (os \subseteq os') \end{array}$$

How a subject's labels may change is critical: conflict of interest must not arise between members of a group. For each group g, we have a representative object denoted $hwm(g)$. The class of this object is initially $(\mathtt{G}\; g\; \{\})$. A request to increase its classification is provided by a family of relabel functions of the form

$$\mathtt{grpaccess} : ORG \nrightarrow Class \nrightarrow Class \nrightarrow Class$$

where $\mathtt{grpaccess}(o)$ defines when group labels may be modified to include organization o. It is specified as

$$\begin{array}{l} \mathtt{grpaccess}(o) \mathrel{\widehat=} \\ \quad \lambda\, s : Class \mid s = (\mathtt{C}\; l\; g\; gs\; os) \bullet \\ \qquad \lambda\, a : Class \mid a = (\mathtt{G}\; g'\; os') \wedge (g = g') \wedge (os \subseteq os') \wedge (os' \cup \{o\} \subseteq ORGNC) \\ \qquad \bullet\; (\mathtt{G}\; g'\; os' \cup \{o\}) \end{array}$$

A subject may consult for an organization if that organization is specified in its group label. A subject classification is upgraded by

$$\mathtt{mbrupgrade} : (Class \times ORG) \nrightarrow Class \nrightarrow Class \nrightarrow Class)$$

where $\mathtt{mbrupgrade}(g, o)$ is used to upgrade a subject member of group g, to include organization o, but only if appropriate. It is defined as

$$\begin{array}{l} \mathtt{mbrupgrade}((\mathtt{G}\; g\; os), o) \mathrel{\widehat=} \\ \quad \lambda\, s : Class \mid (s = (\mathtt{C}\; l'\; g'\; gs'\; os')) \wedge (g = g') \wedge (o \in os) \bullet \\ \qquad \lambda\, a : Class \mid (a = (\mathtt{C}\; l''\; g''\; gs''\; os'')) \wedge (s \leq_c a) \\ \qquad\qquad \bullet\; \mathtt{C}\; l''\; g''\; gs''\; os'' \cup \{o\} \end{array}$$

We assume that appropriate configuration macros create the special group objects. In addition to object relabeling, we assume that the kernel provides a $\mathtt{RelabelS}(s, f)$ operation, which applies the relabeling function f to both the *amin* and *vmax* of subject s. A simple configuration macro can be defined for starting a consultation. Recall that SAPI primitives are presumed atomic.

```
sapi_Consult(s,o):=
  RelabelO(hwm(group(s)), grpaccess(o)(hwm(group(s)));
  RelabelS(s, mbrupgrade(lab(hwm(group(s))), o)).
```

where `group(s)`, gives the group of which subject `s` is a member. The reader should confirm, by checking the definition of function `mbrupgrade`, that both the *amin* and *vmax* can be successfully updated by `RelabelS` above.

A number of observations can be made about this example. First, it illustrates the credibility and degree of trust placed in the configuration macros. An incorrectly specified macro could specify that a subject gets an unexpected classification. Analyzing flow policies and macros for sanity checks, or that they meet some more abstract notion of security, is a topic for future research (one could use the definition of noninterference for aggregation and Chinese Walls from [6] on the macros). A simple analysis that can be done on the Chinese Wall policy is to check for upgrades requested from above. The function `GrpAccess` specifies upgrades from above and thus is a source of a covert channel: a high-level subject can signal down to a low-level subject by choosing to either consult, or not, for a given organization (and affecting the Chinese Wall around the group). Meadows [13] notes that the authors of [9] believe that such channels are too slow to be harmful; however, they fear that the channel may be harmful if the denial is allowed to be temporary, since the high-level subject could then cause denial of access to the same object over and over again. Permitting such upgrades from above provides more flexibility than 'pure' noninterference definitions of such Chinese Walls, which would make it difficult to build a secure system that provided true multilevel organization information. Of course, if we partition groups by sensitivity level, or require that consultants agree to perform consult requests at a low-level, then the covert channel can be eliminated. These variations on the security requirements can be accurately captured by modifying the above flow and relabel policy (which in turn will be an upgrade-from-below policy).

Constructing the lattice and relabel policy, constructing configuration macros, and determining appropriate bindings is an arduous task. We should note that we used some of the techniques in [7] to help design the policies, but for reasons of space we cannot include a description fo them here. We hope to report on these techniques at a later date.

5 Supporting Label Upgrades from Above

Here we consider how to approach implementing a restricted class of flow and relabel policies, in particular, a class of upgrade policies.

A security policy composed of lattice $(L, \leq)$ and relabeling policy R is an upgrade-from-below policy if for all relabel functions $f \in R$. Then

$$\forall\, s : \operatorname{dom} f \bullet (\forall\, a : \operatorname{dom}(f(s)) \bullet s \leq a \leq f(s)(a))$$

Providing secure system support for such policies is (relatively) straightforward to implement and examples can be found in [15, 18]. However, security policies may require upgrades from above. Consider the mark-for-upgrade policy. Suppose that objects are marked (for upgrading), when a high-level subject (read) accesses the object, where reading an object is interpreted as a desire for future access. A naive implementation of this policy is not secure, if a high-level subject can modify a low-level label, with the change visible to a low-level subject. We consider the case where such a label change is not visible to the low-level user.

5.1 Secure Canonical Upgrade Policies

Define a family of equivalence relations on security classes as $(\_ \approx_{\_} \_) : L \to (L \leftrightarrow L)$, where given classes $l, a, b \in L$, then $a \approx_l b$ is interpreted to mean that a subject at class l may not distinguish (by just examining the classes) between security labels a and b.

Example 4 An implementation might use a canonical label scheme to represent equivalence classes, and it is the canonical labels that users see bound to objects and subjects. For example, for the mark-for-upgrade policy in Example 3, define $\texttt{lo} \approx_{\texttt{mlo}} \texttt{mlo}$, but $(\texttt{lo} \not\approx_{\texttt{hi}} \texttt{mlo})$. In an implementation, a low-level subject cannot see a low-level object getting marked by a high-level subject (regard `lo` as the canonical label for `lo` and `mlo`, when viewed from the low-level). △

The system and policy must have some additional properties. We assume that the label view equivalence relation $\approx_l$ is an equivalence relation for each view class $l : L$. We further assume that canonical security classes are symmetric, in that

$$\forall\, a, b : L \bullet a \approx_a b \Leftrightarrow a \approx_b b \qquad \text{[SCUP1]}$$

A relabeling, whether requested from above or below, may result only in an upgrade.

$$\begin{array}{l} \forall\, f : R;\ s : L \bullet s \in \operatorname{dom} f \Rightarrow \\ \quad \forall\, a : \operatorname{dom} f(s) \bullet a \leq f(s)(a) \end{array} \qquad \text{[SCUP2]}$$

The final requirement for a secure canonical upgrade policy is that no sequence of relabeling function application can lead a low-level subject to deduce anything about high-level relabel requests. This is a noninterference-style property (but at the level of the policy, not the system). Starting at any label a we want to ensure that for every sequence of relabel requests applied to a, then the high-level requests in that sequence may not visibly affect the resulting label (from the perspective of a low-level subject).

A sequence of relabel requests is represented as a sequence of values from $R \times L$, where (f, s) represents relabel function f invoked at level s. The final value, after the application of relabeling sequence $t \in \operatorname{seq}(R \times L)$, to a label $a : L$ is defined by $\mathit{relabel}(t, a)$, where

$$\begin{array}{rcl} \mathit{relabel}(\langle\rangle, a) & \widehat{=} & a \\ \mathit{relabel}(t \frown \langle (f, s) \rangle, a) & \widehat{=} & \text{let } b = \mathit{relabel}(t, a) \text{ in} \\ & & \text{if } b \in \operatorname{dom} f(s) \text{ then } f(s)(b) \text{ else } b \end{array}$$

Given a sequence of relabel requests t and a class l, then the *projection* of t through l, denoted $t \upharpoonright l$, gives the sequence t, but with all those requests made at classes that are not dominated by l removed.

$$\begin{array}{rcl} \langle\rangle \upharpoonright l & \widehat{=} & \langle\rangle \\ (t \frown \langle (f, s) \rangle) \upharpoonright l & \widehat{=} & \text{if } s \not\leq l \text{ then } (t \upharpoonright l) \text{ else } (t \upharpoonright l) \frown \langle (f, s) \rangle \end{array}$$

Thus, the noninterference policy requirement is simply stated as

$$\begin{array}{l} \forall\, a, l : L;\ t : \operatorname{seq}(R \times L) \bullet \\ \quad \mathit{relabel}(t, a) \approx_l \mathit{relabel}(t \upharpoonright l, a) \end{array} \qquad \text{[SCUP3]}$$

Finally, we define a *secure canonical upgrade policy* to be a policy $((L, \leq), R, \approx)$ that satisfies the above properties.

Lemma 1 Every upgrade-from-below policy is a secure canonical upgrade policy.
Proof: Follows if the label equivalence relation is defined by $a \approx_l b \Leftrightarrow a = b$ $\square$

Theorem 1 (Unwinding) A security policy $((L, \leq), R, \approx)$ upholds condition SCUP3 if

- No relabeling function, when requested from a high-level, may perform a relabeling that is visible at a low-level class.

$$\begin{aligned} &\forall s : L;\ f : R \bullet s \in \operatorname{dom} f \Rightarrow \\ &\quad \forall a : \operatorname{dom} f(s), l : L \bullet \\ &\qquad (s \not\leq l) \Rightarrow a \approx_l f(s)(a) \end{aligned} \qquad \text{[UW1]}$$

- Relabeling functions must be consistent for canonical labels, that is, a relabeling function should map equivalent classes to equivalent classes.

$$\begin{aligned} &\forall s : L;\ f : R \bullet s \in \operatorname{dom} f \Rightarrow \\ &\quad \forall a, b : \operatorname{dom} f(s);\ l : L \bullet \\ &\qquad a \approx_l b \Rightarrow f(s)(a) \approx_l f(s)(b) \end{aligned} \qquad \text{[UW2]}$$

Proof: Use induction on the length of the trace t. The theorem trivially holds for the base case when $t = \langle\rangle$. Assume that the theorem holds for a trace t (inductive hypothesis), and consider the trace $t \frown \langle(f, s)\rangle$, for arbitrary $f : relabel$ and $s : L$ (inductive step).

- If $s \not\leq l$, then if $s \in \operatorname{dom} f$, then the inductive hypothesis and unwinding rule UWN1 above implies that $f(s)(relabel(t, a)) \approx_l relabel(t, a)$ and it follows from the definition of *relabel* that $relabel(t \frown \langle(f, s)\rangle, a) \approx_l relabel(t, a)$.

  Similarly, in this case, given that $(t \frown \langle(f, s)\rangle) \restriction l = t \restriction l$, we have $relabel((t \frown \langle(f, s)\rangle) \restriction l, a) \approx_l relabel(t, a)$. Since $\approx_l$ is an equivalence relation, we have

  $$s \not\leq l \Rightarrow relabel((t \frown \langle(f, s)\rangle) \restriction l, a) \approx_l relabel((t \frown \langle(f, s)\rangle), a)$$

- If $s \leq l$, then if $s \in \operatorname{dom} f$ and $a, b \in \operatorname{dom} f(s)$, we have by the inductive hypothesis and unwinding rule UWN2 that $f(s)(relabel(t, a)) \approx_l f(s)(relabel(t \restriction l, a))$. But since $x \leq l$ implies $(t \restriction l) \frown \langle(f, s)\rangle = (t \frown \langle(f, s)\rangle) \restriction l$, then it follows from the definition of *relabel* that

  $$s \leq l \Rightarrow relabel((t \frown \langle(f, s)\rangle) \restriction l, a) \approx_l relabel((t \frown \langle(f, s)\rangle), a)$$

Gathering up these two results, the theorem is proven by induction. $\square$

5.2 System Model

The security model proposed in Section 2.2 may be used to to define what is meant for a system to be secure according to a canonical policy, with addition of two new axioms. This revised model supports (secure) relabeling operations for objects. Supporting subject relabeling is a topic for future research.

Axiom 4 Relabel operations may not change subject bindings, that is, for all transitions from state v to v', and subjects $s : S$, then $amin(s) = amin'(s)$ and $vmax(s) = vmax'(s)$. This axiom replaces Axiom 3. $\diamond$

Axiom 5 A subject cannot distinguish equivalent labels by testing for a change in possible accesses.

$$
\begin{array}{l}
\forall\, s : S;\ a, b : L \bullet \\
\quad a \approx_{vmax(s)} b \Rightarrow (a \leq vmax(s)) \Leftrightarrow (b \leq vmax(s)) \wedge \\
\quad a \approx_{amin(s)} b \Rightarrow (amin(s) \leq a) \Leftrightarrow (amin(s) \leq b)
\end{array}
$$

$\diamond$

Note that while we give this requirement as a model axiom, it is really a requirement on subject bindings, that is, part of the abstract security configuration.

Lemma 2 Given a secure canonical relabel policy $(L, \leq, R, \approx)$ enforced by a secure system (according to Axioms 1 through 5), then for all valid states $v, v' \in V$,

$$
\begin{array}{l}
\forall\, s : S \bullet \\
\quad vmax(s) \approx_{vmax(s)} vmax'(s) \Rightarrow vmax(s) = vmax'(s)
\end{array}
$$

Proof: Given $vmax(s) \approx_{vmax(s)} vmax'(s)$, then the reflexivity of $\leq$ and Axiom 5 implies that $vmax(s) \leq vmax(s) \Leftrightarrow vmax'(s) \leq vmax(s)$, that is, $vmax'(s) \leq vmax(s)$. But SCUP1 implies that $vmax(s) \approx_{vmax'(s)} vmax'(s)$ also holds, and we can similarly prove that this implies that $vmax(s) \leq vmax'(s)$. Thus, by the antisymmetry of $\leq$ we have $vmax(s) = vmax'(s)$.
Corollary: We can similarly prove that

$$
\begin{array}{l}
\forall\, s : S \bullet \\
\quad amin(s) \approx_{amin(s)} amin'(s) \Rightarrow amin(s) = amin'(s)
\end{array}
$$

$\square$

Theorem 2 A system that upholds Axioms 1 through 5 for a secure canonical relabel policy is multilevel secure in the sense that it enforces the read-down/write-up rule, and it is not possible for a high-level or disjoint-level subject to interfere with the potential accesses of a low-level subject.
Proof: Axiom 1 enforces the usual read-down/write-up rule, and SCUP2 only permits upgrading of object labels. In the next section we do a noninterference style analysis on the effect that relabeling operations can have on accesses. $\square$

5.3 Security Analysis

The main purpose of this security analysis is to argue that Axioms 1 through 5 and our characterization of a secure canonical relabel policy are a reasonable basis for multilevel security.

But given the abstract nature of the model, and since we have not defined how operations may affect the nonlabel part of the state (e.g., access control, object creation), it is not possible for us to do a *complete* noninterference analysis on the model. Instead, the analysis is done only on the potential accesses that a subject may make. Furthermore, we will not consider object creation or deletion, and we also assume that every relabel transition operation in T relabels is *step-consistent* in the sense that the modification of a particular subject or object label depends only on the current label of the subject or object and the transition operator; it does not depend on any other part of the state of the system. This is not an unreasonable assumption.

The potential accesses of a subject s are determined by the objects the subject may access on the basis of $vmax(s)$ and $amin(s)$. Two states v, v' look the same to subjects dominated by class $l : L$ if $v \equiv_l v'$, where

$$\begin{aligned}
& v \equiv_l v' \mathrel{\widehat=} \\
& \quad \forall\, x : S;\ o : O \bullet vmax(x) \leq l \vee vmax'(x) \leq l \Rightarrow \\
& \qquad amin(x) = amin'(x) \wedge vmax(x) = vmax'(x) \wedge \\
& \qquad lab(o) \approx_{vmax(x)} lab'(o) \wedge \\
& \qquad lab(o) \approx_{amin(x)} lab'(o)
\end{aligned}$$

Thus, if $v \equiv_l v'$, then no subject dominated by l can distinguish between its own bindings ($vmax(x) \approx_{vmax(x)} vmax'(x)$, that is, $vmax(x) = vmax'(x)$, by Lemma 1), nor can the subject distinguish between object classifications. In addition, if $v \equiv_l v'$, then Axiom 5 implies that no subject dominated by l can determine any difference between v and v' by testing for changes in possible accesses to objects, that is,

$$\begin{aligned}
& v \equiv_l v' \Rightarrow \\
& \quad \forall\, x : S;\ o : O \bullet vmax(x) \leq l \vee vmax'(x) \leq l \Rightarrow \\
& \qquad lab(o) \leq vmax(x) \Leftrightarrow lab'(o) \leq vmax'(x) \wedge \\
& \qquad amin(x) \leq lab(o) \Leftrightarrow amin'(x) \leq lab'(o)
\end{aligned}$$

We can now formalize Theorem 2 in terms of an unwound version of noninterference based on the above view-equivalence relation. High-level or disjoint-level (step-consistent) relabel operations cannot interfere with the potential accesses of low-level subjects if

- For any subject s and operation op and states v and $v' = \Sigma(s, op, v)$, then

  $$\forall\, l : L \bullet amin(s) \not\leq l \Rightarrow v \equiv_l v' \qquad \text{[NI1]}$$

  That is, a relabel operation op requested by a high-level subject s cannot alter any part of the state that is visible to a low-level or disjoint-level user.

- For any subject s and operation op and states v, $v' = \Sigma(s, op, v)$, r, and $r' = \Sigma(s, op, r)$, then

  $$\forall\, l : L \bullet amin(s) \leq l \Rightarrow v \equiv_l r \Rightarrow v' \equiv_l r' \qquad \text{[NI2]}$$

  That is, changes that result from a particular transition may depend only on those parts of the state that are visible to the requesting subject.

Proof of NI1: Given any requesting subject s, operation op and states v and $v' = \Sigma(s, op, v)$, and any $o : O$, $x : S$ and $l : L$ such that $amin(s) \not\leq l$ and $vmax(x) \leq l$, then

- If object label $lab(o)$ does not change, then it follows that $lab(o) \approx_{vmax(x)} lab'(o)$.
- If object label $lab(o)$ does change, by Axiom 2 we have $lab'(o) = op(amin(s))(lab(o))$. But given that $amin(s) \not\leq l$, we have by transitivity of $\leq$, that $amin(s) \not\leq vmax(x)$ for any x with $vmax(x) \leq l$. Thus Axiom 2 gives us $lab(o) \approx_{vmax(x)} op(amin(s))(lab(o))$, that is, $lab(o) \approx_{vmax(x)} lab'(o)$.
- Given $amin(s) \not\leq vmax(x)$ and since $amin(x) \leq vmax(x)$, we also have $amin(s) \not\leq amin(x)$. Thus, we can prove in manner similar to the above that $lab(o) \approx_{amin(x)} lab'(o)$.

Axiom 4 implies that $amin(x)$ and $vmax(x)$ do not change. Thus, gathering up these results we have $v \equiv_l v'$ and NI1 is proven.

Proof of NI2: Given requesting subject s, operation op and states v, $v' = \Sigma(s, op, v)$, r, and $r' = \Sigma(s, op, r)$, and any $o : O$, $x : X$ and $l : L$ such that $amin(s) \leq l$ and $vmax(x) \leq l$, then

- Our assumption of step consistency for relabel operations requires that if $op(s)$ changes a particular object label in one state v, then it should also change it in another state r. Thus, if $lab^v(o)$ (denotes the value of $lab(o)$ in state v) is unchanged by operation $op(s)$, then lab^r is also unchanged. Given $v \equiv_l r$, then we have $lab^v(o) \approx_{vmax(x)} lab^r(o)$, and if there is no change it follows that $lab^{v'}(o) \approx_{vmax(x)} lab^{r'}(o)$ and similarly $lab^{v'}(o) \approx_{amin(x)} lab^{r'}(o)$
- If $lab^v(o)$ changes, then by step consistency so does $lab^r(o)$. We have by Axiom 2

  $$lab^{v'}(o) = op(amin^v(s))(lab^v(o))$$
  $$lab^{r'}(o) = op(amin^r(s))(lab^r(o))$$

  Since $amin^v(s) \leq l$ and $v \equiv_l r$, then $amin^v(s) = amin^r(s)$. We also have $lab^v(o) \approx_{vmax^v(x)} lab^r(o)$. Thus, by UWN2 we have $lab^{v'}(o) \approx_{vmax^v(x)} lab^{r'}(o)$, and thus by Axiom 4, we have $lab^{v'}(o) \approx_{vmax^{v'}(x)} lab^{r'}(o)$.
- Since $amin^v(x) \leq vmax^v(x)$, we can similarly prove that $lab^{v'}(o) \approx_{amin^{v'}(x)} lab^{r'}(o)$.

Axiom 4 implies that $amin^v(x)$ and $vmax^r(x)$ do not change. Thus, gathering up these results, we have $v' \equiv_l r'$ and NI2 is proven.

6 Mark-for-Upgrade Policies Revisited

The mark-for-upgrade policy described in Section 4.1 can be modified to permit upgrades from above, assuming that we have support for canonical labels. Intuitively, two labels are equivalent from some class l if their effective levels are the same and those upgrade marks that may be viewed from l are also the same. The `mark` relabel function could be simply specified as

$$\mathtt{mark} \mathrel{\widehat{=}} \lambda(s : \mathit{labels}) \bullet \lambda(o : \mathit{labels}) \bullet \mathit{insert}(o, \mathit{eff}(s))$$

where `mark` defines that it is the effective level of the requester that is used to mark any object's classification. This permits upgrades from above.

Given the policy in Figure 2, suppose an object at class $(\mathtt{C}\,\mathtt{a})$ was marked by a subject with effective class $\mathtt{b}$, giving it the new label $(\mathtt{M}\,\mathtt{b}\,(\mathtt{C}\,\mathtt{a}))$. This object may still be read by a subject 'cleared' to $\mathtt{c}$ $(\mathit{vmax} = (\mathtt{M}\,\mathtt{d}\,(\mathtt{C}\,\mathtt{c})))$, but given the definition of *insert*, this subject cannot mark the object (upgrade paths are total). This is the source of a covert channel: if the subject with effective class $\mathtt{b}$ decides *not* to mark the object, then the $\mathtt{c}$ cleared subject can mark it, giving it label $(\mathtt{M}\,\mathtt{c}\,(\mathtt{C}\,\mathtt{a}))$, which, from the $\mathtt{c}$ subject's view is a different label from $(\mathtt{M}\,\mathtt{b}\,(\mathtt{C}\,\mathtt{a}))$. This is a flaw in the policy specification and it can be easily found by applying the flow analysis, that is, it is not a secure canonical label policy. This covert channel does not exist for the `mark` function if the base lattice is a total ordering: every insertion into the upgrade path is possible. Note that, even if subjects may distinguish only the effective class of labels, the policy is still not secure (in this case, the `mark` function fails the second unwinding condition).

For the purposes of this paper we consider only mark for *read-only* upgrade, that is, marked objects get upgraded to read-only, and may not be subsequently marked or upgraded. This is in contrast with the (mark-from-below) policy presented in Section 4.1, which permits a chain of upgrades. It is possible to generalize the policy we describe here to permit a series of upgrade paths, when no covert channels can arise.

6.1 A Secure Canonical Upgrade Policy

Since we are concerned with only mark for read-only upgrade, the proposed policy is somewhat simpler than the original policy in Section 4.1. Given a base lattice $(L, \leq)$, the security classes for the new policy are labels of the form (a, A, A'), where $a : L$ represents the effective class of the information, $A : \mathbb{P}\, L$ defines (effective) classes to which the label may flow, and $A' : \mathbb{P}\, L$ represents security classes for which the label has been marked. For example, given the base lattice in Figure 2, label $(\mathtt{a}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}, \{\mathtt{b}\})$ represents information with effective class $\mathtt{a}$ which has been marked by security class $\mathtt{b}$. When 'deleted' by security class $\mathtt{a}$, the label should change to $(\mathtt{a}, \{\mathtt{b}\}, \{\mathtt{b}\})$, representing 'read-only' information with an effective class of $\mathtt{a}$, which may flow only to security class $\mathtt{b}$. Had the original label also been marked by security class $\mathtt{c}$, then after 'deletion' the resulting label would be $(\mathtt{a}, \{\mathtt{b}, \mathtt{c}\}, \{\mathtt{b}\})$.

A label (a, A, A') is valid if, for all $b \in A$ then $a \leq b$, which is reasonable given that A defines classes that a may flow to. Define *Class* to be the set of all valid labels (a, A, A'). The ordering relation between these labels forms a lattice and is defined as

$$(a, A, A') \leq (b, B, B') \mathrel{\widehat{=}} (a \leq b) \wedge (A \supseteq A') \wedge (B \subseteq B')$$

Given $a : L$, define $\lceil a \rceil = \{b : L \mid a \leq b\}$ and $\lfloor a \rfloor = \{b : L \mid b \leq b\}$. These are order-preserving mappings [3], in the sense that given $a, b \in L$, then $a \leq b \Leftrightarrow \lceil a \rceil \supseteq \lceil b \rceil \Leftrightarrow \lfloor a \rfloor \subseteq \lfloor b \rfloor$. Thus the new lattice policy preserves the orderings of the base lattice in the sense that $(a \leq b) \Leftrightarrow (a, \lceil a \rceil, \lfloor a \rfloor) \leq (b, \lceil b \rceil, \lfloor b \rfloor)$.

An object that is initially 'bound' to class $a : L$ is actually bound to label $(a, \lceil a \rceil, \{\})$. An object with binding $(a, \lceil a \rceil, A)$, where $A \subseteq \lceil a \rceil$, is interpreted as an object marked by classes in A. An object with binding (a, A, A), with $A \subseteq \lceil a \rceil$ and $a \notin A$ is interpreted as deleted, but read-only to classes in A.

A subject with an 'intended' $vmax(s)$ of class $a : L$, should have $vmax(s) = (a, \{a\}, L)$. This ensures that the subject may view an object, regardless of the object's mark-for-upgrade set. A subject with an 'intended' $amin(s)$ of class $a : L$, should have $amin(s) = (a, \lceil a \rceil, \{\})$. Again, this ensures that the subject may alter an object, regardless of the object's mark-for-upgrade set.

A subject with ($amin$) class $(s, \lceil s \rceil, \{\})$ may mark an object that has not been deleted if the effective class of the subject strictly dominates the effective class of the object. Define the partial relabel function `mark`

$$\begin{aligned}
&\mathtt{mark} : \mathit{Class} \nrightarrow \mathit{Class} \nrightarrow \mathit{Class} \\
&\mathtt{mark} \mathrel{\widehat{=}} \lambda(s : L;\ S, S' : \mathbb{P}\, L) \mid S = \lceil s \rceil \wedge S' = \{\} \bullet \\
&\qquad\qquad \lambda(a : L;\ A, A' : \mathbb{P}\, L) \mid A = \lceil a \rceil \wedge A' \subseteq A \wedge a < s \\
&\qquad\qquad\qquad \bullet (a, A, A' \cup \{s\})
\end{aligned}$$

Note the pre-condition on function `mark`. It is defined only if the object has not been 'deleted', and, for consistency, is currently marked only by classes to which it may flow ($A' \subseteq \lceil a \rceil$). For example, a subject with effective class `b` (Figure 2) marking a previously unmarked and not deleted object with effective class `a` results in relabeling

$$\mathtt{mark}((\mathtt{b}, \{\mathtt{b}, \mathtt{d}\}, \{\}))((\mathtt{a}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}, \{\})) = (\mathtt{a}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}, \{\mathtt{b}\})$$

The `mdel` relabel function updates an object label so that it becomes a read-only label, viewable only by those who had previously marked the label. It is defined as

$$\begin{aligned}
&\mathtt{mdel} : \mathit{Class} \nrightarrow \mathit{Class} \nrightarrow \mathit{Class} \\
&\mathtt{mdel} \mathrel{\widehat{=}} \lambda(s : L;\ S, S' : \mathbb{P}\, L) \mid S = \lceil s \rceil \wedge S' = \{\} \bullet \\
&\qquad\qquad \lambda(a : L;\ A, A' : \mathbb{P}\, L) \mid A = \lceil a \rceil \wedge A' \subseteq A \wedge s \leq a \\
&\qquad\qquad\qquad \bullet (a, A', A')
\end{aligned}$$

If a subject (s, S, S') requests (from below) that an object with label $(a, \lceil a \rceil, A')$ be deleted and since an object label may not be marked by its own effective class, we have $a \notin A'$. Thus the new label for the object (a, A', A') is not visible to any subject at security class a, that is, it appears deleted. Note the pre-condition of `mdel` implies that the relabel function may not be applied to an already deleted object. For example, a subject with effective class a (Figure 2) deleting an object with effective class a, previously marked by security classes b and c results in the relabeling

$$\mathtt{mdel}((\mathtt{a},\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\},\{\}))((\mathtt{a},\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\},\{b,c\})) = (\mathtt{a},\{\mathtt{b},\mathtt{c}\},\{\mathtt{b},\mathtt{c}\})$$

An object with label $(a, \lceil a \rceil, \{\})$ has not been marked at all. When deleted with mdel, it is upgraded to class $(a, \{\}, \{\})$, a class that is not visible to any subject except special garbage collectors (running at that class). This mark for deleting strategy (rather than actually deleting) is necessary for the relabel function to satisfy policy unwinding condition UWN2.

The label equivalence relation may now be defined.

$$\begin{aligned}&(a, A, A') \approx_{(c,C,C')} (b, B, B') \mathrel{\widehat{=}}\\ &\quad a = b \wedge\\ &\quad A \cap \lfloor c \rfloor = B \cap \lfloor c \rfloor \wedge\\ &\quad A' \cap \lfloor c \rfloor = B' \cap \lfloor c \rfloor\end{aligned}$$

Note that under this relationship, if classes, (a, A, A') and (b, B, B') appear equivalent when viewed from class (c, C, C'), then those markings visible to c are equal, that is, $A' \cap \lfloor c \rfloor = B' \cap \lfloor c \rfloor$). For example, a subject with an effective *vmax* class of b may not distinguish between an object with effective class a marked by either b, or by b and c, that is,

$$(\mathtt{a},\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\},\{\mathtt{b}\}) \approx_{((\mathtt{b},\{\mathtt{b}\},\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\})} (\mathtt{a},\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\},\{\mathtt{b},\mathtt{c}\})$$

6.2 Security Analysis

Lemma 3 Given valid labels $(a, \lceil a \rceil, \{\})$ and (b, B, B') then

$$a \le b \Leftrightarrow (a, \lceil a \rceil, \{\}) \le (b, B, B')$$

Proof: $\{\} \subseteq B'$ trivially holds. If $a \le b$, then $\lceil b \rceil \subseteq \lceil a \rceil$, and since (b, B, B') is a valid label we have $B \subseteq \lceil b \rceil$, which implies $B \subseteq \lceil a \rceil$. Thus

$$a \le b \Rightarrow (a, \lceil a \rceil, \{\}) \le (b, B, B')$$

and if follows by definition that

$$(a, \lceil a \rceil, \{\}) \le (b, B, B') \Rightarrow a \le b$$

□

Theorem 3 The proposed mark-for-read-only-upgrade policy is secure.
Proof: We assume that the proposed policy will be enforced by a system that has already been proven to uphold Axioms 1 to 4. To prove security for the final configuration, it is necessary to prove that the mark-for-read-only-update policy is a secure conglomerate update policy and that Axiom 5 holds for all subject bindings.

- (SCUP1) It follows from its definition, that $\approx_l$ is an equivalence relation. Given equivalence $(a, A, A') \approx_{(a,A,A')} (b, B, B')$, then $a = b$, which implies that $\lfloor a \rfloor = \lfloor b \rfloor$. Thus, it follows that $(a, A, A') \approx_{(b,B,B')} (b, B, B')$.

- (SCUP2) The `mark` relabel function updates object label $(a, \lceil a \rceil, A')$ to $(a, \lceil a \rceil, A' \cup \{s\})$, and since $A' \subseteq A' \cup \{s\}$, then

  $$(a, \lceil a \rceil, A') \leq \texttt{mark}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A')$$

  The `mdel` relabel function updates object label $(a, \lceil a \rceil, A')$ to (a, A', A'), and its precondition $A' \subseteq \lceil a \rceil$, implies that

  $$(a, \lceil a \rceil, A') \leq \texttt{mdel}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A')$$

- (UWN1) A `mark` operation properly requested by a subject with effective class s upgrades security class $(a, \lceil a \rceil, A')$ to $(a, \lceil a \rceil, A' \cup \{s\})$. Given an arbitrary label (c, C, C') with $(s, \lceil s \rceil, \{\}) \not\leq (c, C, C')$, then it follows by Lemma 3 that $s \not\leq c$, that is, $s \notin \lfloor c \rfloor$. Thus, we have $A' \cap \lfloor c \rfloor = (A' \cup \{s\}) \cap \lfloor c \rfloor$ and

  $$(a, \lceil a \rceil, A') \approx_{(c,C,C')} \texttt{mark}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A')$$

  A `mdel` operation properly requested by a subject with effective class s upgrades security class $(a, \lceil a \rceil, A')$ to (a, A', A'). Given an arbitrary label (c, C, C') with $(s, \lceil s \rceil, \{\}) \not\leq (c, C, C')$, then as above, we have $s \not\leq c$, and since $s \leq a$, then $a \not\leq c$. Thus by transitivity of $\leq$ we have $\lceil a \rceil \cap \lfloor c \rfloor =$, and since the precondition of `mdel` implies $A' \subseteq \lceil a \rceil$, then we also have $A' \cap \lfloor c \rfloor =$. Thus, we have

  $$(a, \lceil a \rceil, A') \approx_{(c,C,C')} \texttt{mdel}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A')$$

- (UWN2) A `mark` operation properly requested upgrades a label $(a, \lceil a \rceil, A')$ to $(a, \lceil a \rceil, A' \cup \{s\})$ and a security class $(b, \lceil b \rceil, B')$ to $(b, \lceil b \rceil, B' \cup \{s\})$. Given $(a, \lceil a \rceil, A') \approx_{(c,C,C')} (b, \lceil b \rceil, B')$, then $a = b$, and $A' \cap \lfloor c \rfloor = B' \cap \lfloor c \rfloor$, which trivially implies that $(A' \cup \{s\}) \cap \lfloor c \rfloor = (B' \cup \{s\}) \cap \lfloor c \rfloor$. Thus we have

  $$\begin{aligned}&(a, \lceil a \rceil, A') \approx_{(c,C,C')} (b, \lceil b \rceil, B') \Rightarrow \\ &\quad \texttt{mark}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A') \approx_{(c,C,C')} \texttt{mark}(s, \lceil s \rceil, \{\})(b, \lceil b \rceil, B')\end{aligned}$$

  A `mdel` operation properly requested upgrades a security class $(a, \lceil a \rceil, A')$ to (a, A', A') and a security class $(b, \lceil b \rceil, B')$ to (b, B', B'). Thus it trivially follows that

  $$\begin{aligned}&(a, \lceil a \rceil, A') \approx_{(c,C,C')} (b, \lceil b \rceil, B') \Rightarrow \\ &\quad \texttt{mdel}(s, \lceil s \rceil, \{\})(a, \lceil a \rceil, A') \approx_{(c,C,C')} \texttt{mdel}(s, \lceil s \rceil, \{\})(b, \lceil b \rceil, B')\end{aligned}$$

- (Axiom 5) Given $(a, A, A') \approx_{vmax(s)} (b, B, B')$, where $vmax(s) = (c, \{c\}, L)$, then

  $$a = b \Rightarrow (a \leq c \Leftrightarrow b \leq c)$$

If $A \cap \lfloor c \rfloor = B \cap \lfloor c \rfloor$, then since $c \in \lfloor c \rfloor$ we have $c \in A \Leftrightarrow c \in B$. Thus

$$A \cap \lfloor c \rfloor = B \cap \lfloor c \rfloor \Rightarrow A \supseteq \{c\} \Leftrightarrow B \supseteq \{c\}$$

Finally, since $\{\} \subseteq A' \Leftrightarrow \{\} \subseteq B'$ trivially holds, we have

$$(a, A, A') \approx_{vmax(s)} (b, B, B') \Rightarrow (a, A, A') \leq vmax(s) \Leftrightarrow (b, B, B') \leq vmax(s)$$

Given $(a, A, A') \approx_{amin(s)} (b, B, B')$, where $amin(s) = (c, \lceil c \rceil, \{\})$, then

$$a = b \Rightarrow (c \leq a \Leftrightarrow c \leq b)$$

and it follows from Lemma 3 that $(c, \lceil c \rceil, \{\}) \leq (a, A, A') \Leftrightarrow (c, \lceil c \rceil, \{\}) \leq (b, B, B')$. Thus, we have

$$(a, A, A') \approx_{amin(s)} (b, B, B') \Rightarrow amin(s) \leq (a, A, A') \Leftrightarrow amin(s) \leq (b, B, B')$$

$\square$

7 Conclusion

We have described a security model for relabeling policies that provides the basis for a tiered approach to verification. Section 5 illustrated the philosophy of our approach: a security kernel, verified to uphold Axioms 1 through 5, can be trusted to enforce any abstract policy that is verified to be a secure canonical upgrade policy. Thus, a kernel, once verified, can have the potential to support a wide range of different security policies. Our approach permits rules about relabeling to be abstractly specified as part of the (easier to reason about) security policy. That such low-level policy details as relabeling could be specified and enforced at an abstract level was proposed by Gong and Qian in [8]. Our model generalizes this work: Section 4.1 illustrates how the effect of the $\sqcup$ operator can be captured as an upgrade-from-below security policy.

The base model proposed in Section 2 does not consider covert channels. But given its easily recognizable MAC origins we make the same observation as Meadows [13], *"The Bell-LaPadula model also does not capture covert channels, and this is actually considered an advantage by many, since it allows one to model a system in its broad outlines and attend to the details later"*. Our model represents a reasonable starting point for a line of research into dynamic labeling policies. This research includes the development of a noninterference characterization of our relabeling policies, study of further properties of relabeling policies that can be supported in a secure manner, and general techniques for the specification and verification of relabel policies. This is ongoing work on which we hope to report on in the future.

Acknowledgments

This work was conducted while Simon Foley was an International Fellow at the Computer Science Laboratory, SRI International. He would like to thank SRI and the Computer Science Laboratory for making this possible.

References

[1] D. E. Bell and L. J. La Padula. Secure computer system: unified exposition and MULTICS. Report ESD-TR-75-306, The MITRE Corporation, March 1976.

[2] E. Bertino, L.V. Mancini, and S. Jajodia. Collecting garbage in multilevel secure object stores. In *Proceedings Symposium on Security and Privacy*, pages 106–120, Oakland, CA, May 1994. IEEE Computer Society.

[3] G. Birkoff. *Lattice Theory*, volume XXV of *American Mathematical Society Colloquium Publications.* American Mathematical Society, 1967.

[4] M. Branstad et al. Trusted mach design issues. In *Proceedings Third Aerospace Computer Security Conference*, 1987.

[5] S.N. Foley. Separation of duty using high water marks. In *Proceedings Computer Security Foundations Workshop*, Franconia, New Hampshire, June 1991. IEEE Computer Society.

[6] S.N. Foley. Aggregation and separation as noninterference properties. *Journal of Computer Security*, 1(2), 1992.

[7] S.N. Foley. Reasoning about confidentiality requirements. In *Proceedings Computer Security Foundations Workshop*, Franconia, NH, June 1994. IEEE Computer Society.

[8] L. Gong and X. Qian. Enriching the expressive power of security labels. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):839–841, October 1995.

[9] J.T. Haigh, R.C. O'Brien, P.D. Stachour, and D.L. Toups. The LDV approach to database security. In *Research Directions in Database Security III.* North-Holland, 1993.

[10] C.L. Harrold. The Terry-Wiseman security policy model and examples of its use. Technical Report 90001, Defense Research Agency, St. Andrew's Road, Malvern, U.K., 1990.

[11] J. McLean. The algebra of security. In *Proceedings 1988 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1988.

[12] C. Meadows. Extending the Brewer Nash model to a multilevel context. In *Proceedings Symposium on Security and Privacy*, Oakland, CA, 1990. IEEE Computer Society.

[13] C. Meadows. Policies for dynamic upgrading. In *Database Security: Status and Prospects, IV.* North Holland, 1991.

[14] R.S. Sandhu. Lattice based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[15] R.S. Sandhu and Sushil Jajodia. Honest databases that can keep secrets. In *Proceedings of the 14th National Computer Security Conference*, Washington, DC, 1991.

[16] I. Sutherland, S. Perlo, and R. Varadarajan. Deducibility security with dynamic level assignments. In *Computer Security Foundations Workshop 2*, Franconia, NH, U.S.A., June 1989. IEEE Computer Society Press.

[17] C. Weissman. Security controls in the ADEPT-50 time-sharing system. In *AFIPS Conference Proceedings*, volume 35, pages 119–133. FJCC, 1969.

[18] S.R. Wiseman. On the problem of security in databases. In *Database Security III: Status and Prospects*. Springer, 1989.

[19] J.P.L. Woodward. Exploiting the dual nature of sensitivity labels. In *Proceedings Symposium on Security and Privacy*, pages 23–30, Oakland, CA, April 1987. IEEE Computer Society.