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 [?], 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.

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 [?, ?], 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 [?] and SERCUS [?], 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 [?, ?, ?] and segregation of duties [?]. 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 [?] 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 [?] and the expressive labels proposed by Gong and Qian [?]. With the former, subjects may be given the right to arbitrarily modify particular object

*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.

labels. In [?] 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 **??** 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 **??** 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 **??**, 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 [?]. 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 **??** considers the problems of supporting more general upgrade policies, where the requests for upgrade come from above. Adopting our tiered verification approach, Section **??** 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 $\texttt{lo} < \texttt{hi}$ provides partial relabeling functions

$$\begin{aligned}\texttt{up} &\mathrel{\widehat{=}} \lambda(s : L) \mid s = \texttt{lo} \bullet (\lambda(a : L) \mid a = \texttt{lo} \bullet \texttt{hi})\\ \texttt{down} &\mathrel{\widehat{=}} \lambda(s : L) \mid s = \texttt{hi} \bullet (\lambda(a : L) \mid a = \texttt{hi} \bullet \texttt{lo})\end{aligned}$$

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

$$\begin{aligned}\texttt{sub} &\mathrel{\widehat{=}} \lambda(s : \mathbb{P} L) \mid s = \{h\} \bullet (\lambda(a : \mathbb{P} L) \mid a = \{h\} \bullet \{d\})\\ \texttt{down} &\mathrel{\widehat{=}} \lambda(s : \mathbb{P} L) \mid s = \{d\} \bullet (\lambda(a : \mathbb{P} L) \mid a = \{d\} \bullet \{\})\end{aligned}$$

$\triangle$

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 **??**.

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 $M : S \times O \rightarrow \mathbb{P} A$ is an access matrix, indicating the rights that subjects currently have on objects, $lab : O \rightarrow L$ 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 [?]. 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 $\mathcal{R} : T \rightarrow R$ 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 $\Sigma : S \times T \times V \rightarrow V$, 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

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

This axiom is simply the partial trust adaptation of the star property and simple security condition. As with the Bell-LaPadula model [?], 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.

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

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 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 **??**,

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

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 **??** 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 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 **??**. Some work already exists on noninterference models for particular dynamic labeling policies, including downgrading [?] and Chinese Walls [?]. Formulating general relabel policies in terms of noninterference is a topic for future research.