# Owned Policies for Information Security

Hubie Chen    Stephen Chong
Department of Computer Science
Cornell University
Ithaca, NY 14853, USA
{hubes,schong}@cs.cornell.edu

## Abstract

*In many systems, items of information have owners associated with them. An owner of an item of information may want the system to enforce a policy that restricts use of that information; we call such a policy an* owned policy*. Owned policies can be used in many contexts, including information flow, access control, and software licensing. In this paper we introduce and study a general framework for owned policies.*

*Relationships between security policies for a given system may be dependent on system aspects that change between or during system execution. As a result, there may be only partial knowledge of the structure of security policies available when analyzing a system statically. We demonstrate that our framework permits static reasoning about owned policies under partial knowledge, and we also exhibit tractability results for the problem of inferring security policies.*

## 1. Introduction

In many systems, items of information have owners associated with them. An owner of some item of information may want the system to enforce a policy that restricts use of that information. We call such a policy an *owned policy* [16, 11]. In this paper, we introduce and study a general framework for specifying, reasoning about, and inferring owned policies.

To illustrate the diversity and pervasiveness of owned policies, we present three brief examples of contexts in which they occur.

- **Information flow.** Static information flow control allows the enforcement of end-to-end security properties, and in particular, the property of *noninterference* [5]. A system is noninterfering if the low security outputs of the system are independent of the high security inputs, and thus, an observer with low security clearance cannot deduce any information about high security data. In a system with mutually distrusting principals, different principals may have different information flow requirements; when information originating from different principals is combined, the information flow restrictions of all principals must be enforced. The owners in this example are the principals, and the policies restrict the flow of information. The decentralized label model [16] is a model of owned information flow policies.
- **Access control.** Consider a system with information resources, such as files, that are owned by principals. Suppose that the owner of a file can specify an access control list—a list of principals who may read the resource. Now consider a procedure that principals may invoke, which accesses one or more resources. Assuming the procedure executes with the authority of only the invoking principal, the access control list for the procedure should be at least as restrictive as the access control lists of the resources. The owners in this example are the principals, and the policies restrict access to information. Owner Retained Access Control [11] is a model of owned access control policies.
- **Software licensing.** Software producers distribute software components under licenses that include restrictions on when the software may be composed with other components to form new software. The GNU General Public License (GPL) [4] is an example of such a software license: software distributed under the GPL may be used as components of other software provided the resultant software also is distributed under the GPL. The

owners in this example are the software producers, and the policies restrict the composition of the software components.

Given the relevance of owned policies to a broad range of systems, a general framework for specifying and reasoning about owned policies is desirable. Such a framework facilitates the study of owned policies in these systems: results proved for the general framework will hold for all instantiations of the framework.

The general framework for owned policies presented and studied in this paper has two features that we wish to highlight:

- **Reasoning with partial knowledge of security policy structure.** Some aspects of a system are subject to change between or during executions of that system. For example, users may leave or join the system, users may have privileges and roles added or revoked, and files may have access permissions changed. The structure of security policies may be dependent on some of these system aspects. For example, if a security policy permits only members of the group $G$ to access information, then the user Alice is permitted to access information only if she is a member of the group $G$; however, Alice may be added or removed from the group between executions of the system.

  Our framework permits static reasoning about owned policies, even when only partial knowledge of the run-time security policy structure is available. Furthermore, one can use the framework to reason using additional knowledge that may be revealed through run-time testing of the security policy structure.

- **Inference of policies.** Program analysis is an important tool for proving systems secure. Program analysis typically requires the explication of the security policies for the information the program manipulates, often by program annotations. The burden on the program developers to write down security policy specifications can be significant, especially for large systems. Inference of security policies can reduce the specification burden on developers, allowing them to elide some of the annotations. This enables both the quicker development of secure systems, and the reduction of annotation "clutter" in programs. Inference is particularly complex when there is only partial knowledge of the run-time security policy structure.

  Our framework for owned policies is amenable to the inference of security policies. We present a number of positive tractability results for the inference problem within our framework.

Our model generalizes the decentralized label model of Myers and Liskov [16], which is a model of owned policies for information flow.

The remainder of the paper is organized as follows. In Section 2 we present the basic syntax and semantics of our owned policy model. Some instantiations of the model are given as examples in Section 3. The structure of the model is presented and discussed in Section 4. In Section 5 we introduce mechanisms to reason about owned policies with only partial knowledge of the structure of security policies. Section 6 introduces the problem of security policy inference, and presents a number of positive tractability results for instances of this inference problem within our framework. Finally, in Sections 7 and 8, we discuss related work and conclude.

## 2. Owned Policy Model

In this section we present first the syntax and then the semantics of our model for owned policies.

### 2.1. Syntax

Throughout, $O$ will denote a finite set of *owners*. An *owner hierarchy* is a relation $\geq_o \subseteq O \times O$. An owner hierarchy models an "acts for" relation: if $o_1 \geq_o o_2$, then the owner $o_1$ is able to act on behalf of the owner $o_2$. We assume that $\geq_o$ is reflexive and transitive, that is, $\geq_o$ is a preorder. Owners typically represent principals in a system, but they can also be used to model groups and roles, through the use of the acts-for relation [15]; for example, given a group $G$, a special owner $o_G$ represents that group, and all principals that are members of the group $G$ are able to act for $o_G$. The acts-for relation is similar to the *speaks for* relation [9].

Throughout, $P$ will denote a finite set of *policies* for specifying information use. A *policy hierarchy* is a relation $\geq_p \subseteq P \times P$. A policy hierarchy models an "at least as restrictive" relation: when $p_1 \geq_p p_2$, a value with policy $p_2$ can be used anywhere that a value with policy $p_1$ could be. We also assume that $\geq_p$ is reflexive and transitive, that is, $\geq_p$ is a preorder.

An owner of an item of data may specify a policy that she requests to be enforced on that data; such a specification forms an owned policy. Formally, an *owned policy* is a pair consisting of an owner $o \in O$ and a policy $p \in P$, and is denoted $o{:}p$. We use the capital letters $I$ and $J$ to denote owned policies. The operator $\mathbf{o}(\cdot)$ returns the owner of an owned policy; the operator $\mathbf{p}(\cdot)$ returns the policy of an owned policy. Thus, $\mathbf{o}(o{:}p) = o$ and $\mathbf{p}(o{:}p) = p$.

A *label* is a set of owned policies, denoted $\{o_1{:}p_1,\ \ldots,\ o_n{:}p_n\}$, where each $o_i{:}p_i$ is an owned policy in the label. We use the letter $L$ to denote a single label, and we use $\mathcal{L}$ to denote a class of labels where all labels are over the same owner set $O$ and policy set $P$. For a fixed owner set $O$ and policy set $P$, we denote the class of all labels over $O$ and $P$ by $\mathcal{L}^{\text{all}}$.

Labels can be associated with data. Intuitively, if a label $\{o_1{:}p_1,\ \ldots,\ o_n{:}p_n\}$ is associated with some item of data, then each owner $o_i$ has specified that uses of the data should respect the policy $p_i$, or a more restrictive policy; a system that enforces the label should ensure that all owners' specifications are met. If an owner $o$ does not own a policy in a label, then the label is interpreted as if $o$ does not place any restrictions on the use of the data, as will be seen in Section 2.2. The model is agnostic as to the origin of labels. In particular, whether a given owner $o \in O$ is allowed to specify a policy on an item of data is outside the scope of the model.

A *hierarchy* $H = (\geq_o, \geq_p)$ is a pair consisting of an owner hierarchy and a policy hierarchy. We use $\geq_o^H$ and $\geq_p^H$ to denote the owner hierarchy and policy hierarchy of $H$, respectively. We use $\mathcal{H}$ to denote a class of hierarchies, where all the owner hierarchies are over the same owner set $O$, and all the policy hierarchies are over the same policy set $P$.

**Example 2.1:** Let the owner set $O$ be $\{\text{Alice}, \text{Bob}, \text{Chuck}\}$, and let the relation $\geq_o$ be the reflexive transitive closure of $\{(\text{Chuck}, \text{Bob})\}$, that is, $\geq_o$ is the relation $\{(\text{Chuck, Bob}), (\text{Alice, Alice}), (\text{Bob, Bob}), (\text{Chuck, Chuck})\}$. Let the policy set $P$ be $\{\textit{TopSecret}, \textit{Classified}, \textit{Unclassified}\}$, and let $\geq_p$ be the reflexive transitive closure of $\{(\textit{TopSecret}, \textit{Classified}), (\textit{Classified}, \textit{Unclassified})\}$. Intuitively, the policy *TopSecret* is the most restrictive policy, while the policy *Unclassified* is the least restrictive policy, since a value that is *Unclassified* could be used (for example, read, copied, sent, etc.) anywhere a value that is *TopSecret* could be.

In the label $\{\text{Alice}{:}\,\textit{Classified},\ \text{Bob}{:}\,\textit{TopSecret}\}$, Alice specifies that the policy *Classified* should be enforced, while Bob specifies that the policy *TopSecret* should be enforced; Chuck does not specify any policy in this label. ∎

## 2.2. Semantics

We now provide a semantics for labels relative to a hierarchy $H$. We present two possible semantics for labels, using two different operators. The first, the **X** operator, provides a semantics for any possible hierarchy $H$. The second operator, the **Y** operator, defines a more succinct semantics than the **X** operator, but is only applicable to hierarchies with a certain structure; the **Y** operator is defined in terms of the **X** operator.

**2.2.1. X Operator** The **X** operator provides a semantics of a label $L$, with respect to a hierarchy $H$. The semantics of a label $L$ is a set of *permissions*; a permission is a pair $(o, p) \in O \times P$ having the meaning that owner $o$ gives permission for data labeled with $L$ to be used according to policy $p$.

**Definition 2.2:** The **X** operator is defined for every hierarchy $H$ and label $L$ as follows:

$$\mathbf{X}(H, L) = \{(o, p) \mid \forall I \in L.\ \mathbf{o}(I) \geq_o^H o \Rightarrow p \geq_p^H \mathbf{p}(I)\}$$

∎

While owned policies and permissions are both pairs of owners and policies, we use two different terms and different notation to emphasize the different contexts in which they are used: owned policies (and labels) are independent of any hierarchy; permissions constitute the interpretation of a label under a specific hierarchy.

Two key properties of the **X** operator highlight how a specific hierarchy is reflected in the interpretation of a label. First, permission $(o, p)$ is contained in $\mathbf{X}(H, L)$ only if every owner who can act for $o$ also permits the data to be used according to policy $p$. Equivalently, if an owner $o$ does not give permission for the data to be used according to some policy $p$, then no owner subordinate to $o$ (that is, no owner whom $o$ can act for) can give permission for the data to be used according to $p$.

**Property 2.3:** For every hierarchy $H$, label $L$, owners $o, o' \in O$ and policy $p \in P$, if $o' \geq_o^H o$ and $(o, p) \in \mathbf{X}(H, L)$, then $(o', p) \in \mathbf{X}(H, L)$.

Second, if permission $(o, p)$ is contained in $\mathbf{X}(H, L)$, then $o$ also permits the data to be used according to policy $p'$, for every policy

$p'$ that is at least as restrictive as $p$. This accords with the intended meaning of labels: if data is labeled $\{o_1{:}p_1,\ \ldots,\ o_n{:}p_n\}$ then each owner $o_i$ allows the data to be used according to policy $p_i$, or a policy more restrictive than $p_i$.

**Property 2.4:** For every hierarchy $H$, label $L$, owner $o \in O$ and policies $p, p' \in P$, if $p' \geq_p^H p$ and $(o,p) \in \mathbf{X}(H,L)$, then $(o,p') \in \mathbf{X}(H,L)$.

**Example 2.5:** Consider Example 2.1, where the owner set $O$ is $\{\text{Alice}, \text{Bob}, \text{Chuck}\}$, $\geq_o$ is the reflexive transitive closure of $\{(\text{Chuck}, \text{Bob})\}$, the policy set $P$ is $\{\textit{TopSecret}, \textit{Classified}, \textit{Unclassified}\}$ and $\geq_p$ is the reflexive transitive closure of $\{(\textit{TopSecret}, \textit{Classified}), (\textit{Classified}, \textit{Unclassified})\}$. Let the hierarchy $H$ be $(\geq_o, \geq_p)$. The semantics of the label $\{\text{Alice:}\,\textit{Classified},\ \text{Bob:}\,\textit{TopSecret}\}$ is the following.

$$\begin{aligned}
&\mathbf{X}(H, \{\text{Alice: }\textit{Classified}, \text{Bob: }\textit{TopSecret}\}) = \\
&\qquad \{(\text{Alice}, \textit{Classified}), (\text{Alice}, \textit{TopSecret}), \\
&\qquad (\text{Bob}, \textit{TopSecret}), \\
&\qquad (\text{Chuck}, \textit{Unclassified}), (\text{Chuck}, \textit{Classified}), \\
&\qquad\qquad (\text{Chuck}, \textit{TopSecret})\}
\end{aligned}$$

Observe that Alice gives permission for every policy at least as restrictive as *Classified*, as per Property 2.4, that is, for both of the policies *Classified* and *TopSecret*. Bob only gives permission for the policy *TopSecret*. Since neither Chuck nor any owner that can act for Chuck specified a policy in the label, Chuck gives permission for every policy. A system that enforced this label on an item of data would have to treat the item of data according to the policy *TopSecret*, since this is the only policy that all owners permit.

Consider now the semantics of the label $\{\text{Alice:}\,\textit{Classified},\ \text{Chuck:}\,\textit{TopSecret}\}$.

$$\begin{aligned}
&\mathbf{X}(H, \{\text{Alice: }\textit{Classified}, \text{Chuck: }\textit{TopSecret}\}) = \\
&\qquad \{(\text{Alice}, \textit{Classified}), (\text{Alice}, \textit{TopSecret}), \\
&\qquad (\text{Bob}, \textit{TopSecret}), \\
&\qquad (\text{Chuck}, \textit{TopSecret})\}
\end{aligned}$$

Even though Bob did not specify a policy in the label, since Chuck acts for Bob and Chuck only gives permission for policy *TopSecret*, Bob only gives permission for the policy *TopSecret*, as per Property 2.3. ▮

**2.2.2. Y Operator** For a given hierarchy $H$, if we assume some structure on the preorder $\geq_p^H$, then the semantics of label $L$ (that is, $\mathbf{X}(H,L)$) has some useful additional structure. In particular, if the policy hierarchy $\geq_p^H$ has greatest lower bounds then for any label $L$ and for every owner $o$, there is a least restrictive policy $p$ such that $(o,p) \in \mathbf{X}(H,L)$.

**Definition 2.6:** A hierarchy $H$ is a *meet hierarchy* if $\geq_p^H$ has greatest lower bounds, that is, for any $p_1, p_2 \in P$ there exists a $q \in P$ such that $p_1 \geq_p^H q$ and $p_2 \geq_p^H q$; and for any $q' \in P$, if $p_1 \geq_p^H q'$ and $p_2 \geq_p^H q'$ then $q \geq_p^H q'$.[1] ▮

**Property 2.7:** If $H$ is a meet hierarchy, then for any label $L$ and any owner $o \in O$, there exists a $p \in P$ such that $(o,p) \in \mathbf{X}(H,L)$ and for any $(o,p') \in \mathbf{X}(H,L)$ it is the case that $p' \geq_p^H p$.

We can use Property 2.7 to define a useful alternative semantics for meet hierarchies that captures the same information as the $\mathbf{X}$ operator, but in a more succinct form. The semantics, instead of being sets of permissions, are functions from owners to their corresponding least restrictive policy.

**Definition 2.8:** For a meet hierarchy $H$, and a subset $Q$ of $P$, the *infimum* of $Q$, denoted $\inf_H Q$, is the greatest lower bound in $\geq_p^H$ of all elements in $Q$. ▮

**Definition 2.9:** The $\mathbf{Y}$ operator is defined for every meet hierarchy $H$ and label $L$ as the function from $O$ to $P$ such that

$$\begin{aligned}
\mathbf{Y}(H,L)(o) &= \inf_H \{p \mid (o,p) \in \mathbf{X}(H,L)\} \\
&= \inf_H \{p \mid \forall I \in L.\ \mathbf{o}(I) \geq_o^H o \\
&\qquad\qquad \Rightarrow p \geq_p^H \mathbf{p}(I)\}
\end{aligned}$$

▮

**Example 2.10:** The semantics of the labels in Example 2.5 can be expressed by the $\mathbf{Y}$ operator.

$$\begin{aligned}
&\mathbf{Y}(H, \{\text{Alice: }\textit{Classified}, \text{Bob: }\textit{TopSecret}\}) = \\
&\qquad \begin{cases} \text{Alice} & \mapsto \textit{Classified} \\ \text{Bob} & \mapsto \textit{TopSecret} \\ \text{Chuck} & \mapsto \textit{Unclassified} \end{cases} \\
&\mathbf{Y}(H, \{\text{Alice: }\textit{Classified}, \text{Chuck: }\textit{TopSecret}\}) = \\
&\qquad \begin{cases} \text{Alice} & \mapsto \textit{Classified} \\ \text{Bob} & \mapsto \textit{TopSecret} \\ \text{Chuck} & \mapsto \textit{TopSecret} \end{cases}
\end{aligned}$$

▮

1 Note that if $\geq_p^H$ has least upper bounds and a bottom element, it can be shown that $\geq_p^H$ has greatest lower bounds, where *least upper bounds* is defined analogously to greatest lower bounds. By a *bottom element*, we mean an element $\bot \in P$ such that for all $p \in P$, $p \geq_p^H \bot$.

Note that for meet hierarchies, the hierarchies on which the $\mathbf{Y}$ operator is defined, the semantics provided by the $\mathbf{X}$ and $\mathbf{Y}$ operators are the same. Given the semantics for a label $L$ relative to a meet hierarchy $H$ under one of the operators, it is easy to compute the semantics under the other operator. Indeed, the $\mathbf{Y}$ operator is defined in terms of the $\mathbf{X}$ operator, and it is possible to define the $\mathbf{X}$ operator in terms of the $\mathbf{Y}$ operator: $\mathbf{X}(H, L) = \{(o, p) | o \in O, p \geq_p^H \mathbf{Y}(H, L)(o)\}$.

## 3. Examples

In this section we present some instantiations of our owned policy model.

### 3.1. Decentralized Label Model

In the decentralized label model of Myers and Liskov [15, 16, 14], labels specify restrictions on how information may flow. Principals may own data, and in a decentralized label for an item of data, an owning principal may specify policies, where a policy is a set of principals, called a "reader set". When a principal $o$ specifies the set $R$ as a reader set on a data item, then $o$ requires that the data item be read only by principals contained in the set $R$. There is a reflexive and transitive "acts for" relation $\succeq$ on principals, and if some principal $o$ is permitted to read some data, then all principals who can act for $o$ are implicitly permitted to read the data too.

We can obtain the decentralized label model as a particular instantiation of our model, in the following way. Let the owner set $O$ be the set of principals and let the set of policies $P$ be the power set of $O$. Let $\geq_o$ be the acts-for relation $\succeq$. The relation $\geq_p$ on the policy set $P$ is also defined in terms of $\succeq$: for two reader sets of principals $R_1, R_2 \in P$, define $R_1 \geq_p R_2$ if and only if for all owners $o$ in $R_1$ there is some owner $o'$ in $R_2$ such that $o$ can act for $o'$, that is $o \succeq o'$. In other words, the reader set $R_1$ is "at least as restrictive" as the reader set $R_2$ if every principal permitted to read by $R_1$ is also permitted to read by $R_2$. Our $\mathbf{X}(\cdot, \cdot)$ semantic operator, when specialized to hierarchies induced by an acts-for relation $\succeq$ (as above), is equivalent to the semantic operator of the decentralized label model.[2]

The hierarchy $(\geq_o, \geq_p)$ is a meet hierarchy; the greatest lower bound of two reader sets $R_1$ and $R_2$ is $R_1 \cup R_2$, since the principals who are either in $R_1 \cup R_2$ or can act for a principal in $R_1 \cup R_2$ are exactly the principals who are either in, or can act for, a principal in $R_1$ or $R_2$. As such, the $\mathbf{Y}$ operator is well-defined for the hierarchy $(\geq_o, \geq_p)$.

The semantics of a label is a set of permissions, that is, a set of pairs $(o, R)$, each with an owner $o$ and a reader set $R$. The semantics of a label can be used to determine which principals can read data: if a data item is labeled $\{o_1 : R_1, \ldots, o_n : R_n\}$, then a principal $r$ is permitted to read that data item provided all principals $o$ are prepared to allow $r$ to read the data item, that is, for every principal $o$ there is a reader set $R$ with $r \in R$ such that $(o, R) \in \mathbf{X}((\geq_o, \geq_p), \{o_1 : R_1, \ldots, o_n : R_n\})$. Equivalently, principal $r$ is permitted to read the data item if for every principal $o$, the principal $r$ is contained in the reader set $\mathbf{Y}((\geq_o, \geq_p), \{o_1 : R_1, \ldots, o_n : R_n\})(o)$.

### 3.2. Access Control

A model similar to the decentralized label model just described can be used to model access control lists. Let $O$ be a set of principals, with an acts-for relation $\succeq$ (over $O$). The policy set $P$ consists of pairs of sets of principals: $P = \{(a, d) \mid a, d \subseteq O\}$. In the policy $(a, d)$, the set $a$ is the set of principals that are allowed access, and the set $d$ is the set of principals that are denied access. If owner $o$ can act for owner $o'$, and $o'$ is allowed access ($o' \in a$), then $o$ is implicitly also allowed access; however, if $o$ is denied access ($o \in d$), then $o'$ is implicitly also denied access.

Let $\geq_o$ be the acts-for relation $\succeq$. The relation $\geq_p$ on the policy set $P$ is also defined in terms of $\succeq$: for two access control lists $(a, d)$ and $(a', d')$, define $(a, d) \geq_p (a', d')$ if and only if for all owners $o$ in $a$ there is some owner $o'$ in $a'$ such that $o$ can act for $o'$, and for all owners $o'$ in $d'$ there is some owner $o$ in $d$ such that $o$ can act for $o'$. In other words, the policy $(a, d)$ is "at least as restrictive" as $(a', d')$ if every owner that $(a, d)$ allows access to is also allowed access by $(a', d')$, and every owner that is denied access by $(a', d')$ is also denied access by $(a, d)$.

The semantics of a given label can be used to determine if a principal is allowed access, denied access, or if access for that principal is undetermined by the label. We omit the precise details, as they are slightly too involved for our current purposes. Roughly speaking, a principal $r$ is denied access if some owner denies $r$ access; $r$ is allowed access if $r$ is not denied access and all owners allow $r$ access; otherwise, $r$'s access is undetermined.

2 In some versions of the decentralized label model [14], the owner of a policy is also implicitly a reader. For simplicity, we ignore this detail, although extensions to our model permit us to precisely capture Myers and Liskov's model.

### 3.3. No Owner Acts For Another

A natural restriction of the owned policy model is obtained by not allowing any owner to act for another owner – that is, by requiring the "acts for" relation $\geq_o$ to be the equality relation on $O$, which we denote by $=_O$. In the decentralized label model presented above, this requirement corresponds to allowing principals to act for other principals only in their capacity as readers, and not in their capacity as owners of policies.

In this restricted version of the model, when the policy hierarchy $\geq_p$ of a hierarchy $H$ having the form $(=_O, \geq_p)$ has a least upper bound operation $\sqcup$, every label $L \in \mathcal{L}^{\text{all}}$ is equivalent in $H$ to a label in which each owner owns at most one policy. In particular, if the label $L$ is the set of owned policies $\{o{:}p_1, o{:}p_2, \ldots, o{:}p_n\} \cup L'$, then the semantics of $L$ is equal to the semantics of the label $\{o{:}(p_1 \sqcup \ldots \sqcup p_n)\} \cup L'$. (This is apparent by examining the definition of the $\mathbf{X}$ operator.) In light of this property, it is natural to define a new class of labels, $\mathcal{L}^{\text{own}-\text{one}}$, containing those labels in which each owner owns at most one policy. The class $\mathcal{L}^{\text{own}-\text{one}}$ is much smaller than (and properly contained within) the class $\mathcal{L}^{\text{all}}$.

## 4. Structure of Labels

Our framework for owned policies makes few assumptions on the owner and policy hierarchies, only requiring that they be preorders. Nonetheless, under any hierarchy it can be shown that the class of labels form a lattice structure. In this section, we explain how this structure arises.

For a fixed set of owners $O$ and policies $P$, given any hierarchy $H$ we can define a preorder $\sqsubseteq_H$ on the class of all labels, $\mathcal{L}^{\text{all}}$. Intuitively, if $L_1 \sqsubseteq_H L_2$, then label $L_1$ is no more restrictive than $L_2$ under hierarchy $H$: if owner $o$ gives permission for policy $p$ in $L_2$, then $o$ gives permission for $p$ in $L_1$, that is, $(o, p) \in \mathbf{X}(H, L_2)$ implies $(o, p) \in \mathbf{X}(H, L_1)$.

**Definition 4.1:** For any hierarchy $H$, the preorder $\sqsubseteq_H$ is defined as follows: for any two labels $L_1, L_2 \in \mathcal{L}^{\text{all}}$,

$$L_1 \sqsubseteq_H L_2 \text{ if and only if } \mathbf{X}(H, L_2) \subseteq \mathbf{X}(H, L_1).$$

The equivalence relation $\equiv_H$ is defined as follows: for any two labels $L_1, L_2 \in \mathcal{L}^{\text{all}}$,

$$L_1 \equiv_H L_2 \text{ if and only if } L_1 \sqsubseteq_H L_2 \text{ and } L_2 \sqsubseteq_H L_1.$$

Note that $L_1 \equiv_H L_2$ if and only if $\mathbf{X}(H, L_1) = \mathbf{X}(H, L_2)$. ∎

When $H$ is a meet hierarchy, we can express the preorder $\sqsubseteq_H$ in terms of the $\mathbf{Y}(H, \cdot)$ semantic operator.

**Property 4.2:** When $H$ is a meet hierarchy, $L_1 \sqsubseteq_H L_2$ if and only if for every owner $o \in O$, $\mathbf{Y}(H, L_2)(o) \geq_p^H \mathbf{Y}(H, L_1)(o)$.

Roughly speaking, the class of all labels $\mathcal{L}^{\text{all}}$ form a join semilattice with respect to $\sqsubseteq_H$. Formally, to obtain a join semilattice, we need to identify labels that are equivalent under $\equiv_H$. Let $\mathcal{L}^{\text{all}}/\equiv_H$ denote the quotient set of $\mathcal{L}^{\text{all}}$ with respect to $\equiv_H$, namely $\{[L]_H \mid L \in \mathcal{L}^{\text{all}}\}$, where

$$[L]_H = \{L' \in \mathcal{L}^{\text{all}} \mid L' \equiv_H L\}.$$

The relation $\sqsubseteq_H$ on $\mathcal{L}^{\text{all}}$ induces a partial order on $\mathcal{L}^{\text{all}}/\equiv_H$. We overload the $\sqsubseteq_H$ notation and use it to denote the induced ordering (where for $S_1, S_2 \in \mathcal{L}^{\text{all}}/\equiv_H$, $S_1 \sqsubseteq_H S_2$ if and only if there exist $L_1 \in S_1$ and $L_2 \in S_2$ such that $L_1 \sqsubseteq_H L_2$).

**Theorem 4.3:** *For any hierarchy $H$, the structure $(\mathcal{L}^{\text{all}}/\equiv_H, \sqsubseteq_H)$ is a join semilattice.*

**Proof:** Fix an arbitrary hierarchy $H$. Define the join operation $\sqcup_H$ as follows: for any two labels $L_1, L_2 \in \mathcal{L}^{\text{all}}$, define $[L_1]_H \sqcup_H [L_2]_H$ to be $[L_1 \cup L_2]_H$, where $\cup$ denotes the usual set-theoretic union. It is straightforward to verify that $\mathbf{X}(H, L_1 \cup L_2) = \mathbf{X}(H, L_1) \cap \mathbf{X}(H, L_2)$ for any two labels $L_1, L_2$. Thus, this join operation is well-defined and yields the least upper bound of $[L_1]_H$ and $[L_2]_H$. ∎

Throughout this paper, we will use $\sqcup_H$ to denote the join operation given in the previous proof. We overload the $\sqcup_H$ notation, and also use $L_1 \sqcup_H L_2$ to refer to some label $L$ in the equivalence class $[L_1]_H \sqcup_H [L_2]_H$.

Information can be combined in many possible ways, including by composition (for example, of software components), and computation (for example, the assignment statement $z := x + y$ combines information held in variables $x$ and $y$, and places the result in the variable $z$). Joins of labels arise naturally when combining items of information: the label of the result should be at least as restrictive as the label of each of the constituent inputs. That is, the label of the result should be at least as restrictive as the join of all input labels. Theorem 4.3 demonstrates that for any hierarchy $H$, we can always find a least label satisfying this restrictiveness condition.

The empty label $\{\}$ (that is, the label with no owned policies) is the element of $\mathcal{L}^{\text{all}}$ that is below all other elements with respect to the ordering $\sqsubseteq_H$, since $\mathbf{X}(H, \{\}) = O \times P$. Hence, the join semilattice $\mathcal{L}^{\text{all}}/\equiv_H$ is also a meet semilattice.

**Corollary 4.4:** *For any hierarchy $H$, the structure $(\mathcal{L}^{\text{all}}/\equiv_H, \sqsubseteq_H)$ is a meet semilattice.*

We denote the meet operation on $\mathcal{L}^{\text{all}}/\equiv_H$ with $\sqcap_H$, and overload it to use $L_1 \sqcap_H L_2$ to refer to some label $L$ in the equivalence class $[L_1]_H \sqcap_H [L_2]_H$.

When the hierarchy $H$ is a meet hierarchy, the meet operation $\sqcap_H$ can be described in terms of the $\mathbf{Y}$ operator. Suppose that for a meet hierarchy $H$, the operation $\wedge : P \times P \to P$ gives greatest lower bounds with respect to $\geq_p^H$. The meet of two labels $L_1, L_2 \in \mathcal{L}^{\text{all}}$, that is, $L_1 \sqcap_H L_2$, is equivalent to

$$\{o\colon(\mathbf{Y}(H, L_1)(o) \wedge \mathbf{Y}(H, L_2)(o)) \mid o \in O\}.$$

The meet of the labels has the property that for all owners $o$, the policy $\mathbf{Y}(H, L_1 \sqcap_H L_2)(o)$ is equal to $\mathbf{Y}(H, L_1)(o) \wedge \mathbf{Y}(H, L_2)(o)$.

Meets of labels arise in situations dual to those in which joins arise: if an item of information is used to produce several different results, then the label of that information must not be more restrictive than any of the labels of the results. In summary, joins arise when many labeled inputs are combined to produce a single output, whereas meets arise when one input is used to produce several labeled outputs.

## 5. Run-time Hierarchies

As mentioned in the introduction, when statically analyzing a system it is typically unknown which hierarchy will be in effect at run time. Despite this lack of knowledge, it is desirable to be able to reason statically about the run-time security properties of the system, in particular, to prove that a system will be secure regardless of which hierarchy is in effect at run time. In this section, we show how such static reasoning can be performed using our model.

### 5.1. Classes of Hierarchies

To prove that a system will be secure regardless of which hierarchy is in effect at run time, it suffices to show that the system is secure for all hierarchies that may be in effect at run time. We illustrate this idea with a simple example of pseudo-code where program types are annotated with labels, as in security typed languages. In a security typed language, the type system of the language uses the label annotations to enforce security properties, such as noninterference [5]. One way to achieve noninterference is to ensure that if the value of a variable $x$ depends on the value of a variable $y$, then the label of $x$ is at least as restrictive as the label of $y$.

**Example 5.1:** Consider the following segment of pseudo-code, where program types are annotated with labels over the same owner set $O$ and policy set $P$ as in Example 2.1.

```
1  int{Chuck: TopSecret} x := ...;
2  int{Bob: Classified} y := ...;
3  int{?} z := x + y;
```

In the code above, the type of variable $z$ does not yet have a label, but we would like to determine a suitable label, $L$, for the variable $z$, so that the program is noninterfering.

Since the value of the variable $x$ is used to compute the value assigned to $z$ at line 3, the label of $z$ must be at least as restrictive as the label of $x$, that is, $\{\text{Chuck}\colon \textit{TopSecret}\} \sqsubseteq_H L$, for any hierarchy $H$ that may occur at run time. Similarly, the label of $z$ must be at least as restrictive as the label of $y$, that is, $\{\text{Bob}\colon \textit{Classified}\} \sqsubseteq_H L$. These two requirements on $L$ can be combined into the single requirement $\{\text{Chuck}\colon \textit{TopSecret}\} \sqcup_H \{\text{Bob}\colon \textit{Classified}\} \sqsubseteq_H L$.

Assume that the class of hierarchies that can occur at run time contains two hierarchies: the hierarchy $H_1 = (\geq_o^{H_1}, \geq_p^{H_1})$ which is the same as the hierarchy from Example 2.1 (where Chuck $\geq_o^{H_1}$ Bob and *TopSecret* $\geq_p^{H_1}$ *Classified* $\geq_p^{H_1}$ *Unclassified*); and the hierarchy $H_2 = (\geq_o^{H_2}, \geq_p^{H_2})$, where $\geq_o^{H_2}$ is the equality relation, and $\geq_p^{H_2}$ is the same as $\geq_p^{H_1}$.

If the hierarchy at run time is $H_1$, then $\{\text{Chuck}\colon \textit{TopSecret}\}$ is a suitable label for the variable $z$. On the other hand, if $H_2$ is the hierarchy at run time, then $\{\text{Chuck}\colon \textit{TopSecret}\}$ is *not* a suitable label, since it is not the case that $\{\text{Bob}\colon \textit{Classified}\} \sqsubseteq_{H_2} \{\text{Chuck}\colon \textit{TopSecret}\}$, as the permission (Bob, *Unclassified*) is in the set $X(H_2, \{\text{Chuck}\colon \textit{TopSecret}\})$, but not in the set $X(H_2, \{\text{Bob}\colon \textit{Classified}\})$.

In both hierarchies, however, the label $\{\text{Bob}\colon \textit{Classified}, \text{Chuck}\colon \textit{TopSecret}\}$ is suitable, since for $H \in \{H_1, H_2\}$ we have

$$\{\text{Chuck}\colon \textit{TopSecret}\} \sqcup_H \{\text{Bob}\colon \textit{Classified}\} \sqsubseteq_H \{\text{Bob}\colon \textit{Classified}, \text{Chuck}\colon \textit{TopSecret}\}$$

∎

In the preceding example, the label $\{\text{Bob}\colon \textit{Classified}, \text{Chuck}\colon \textit{TopSecret}\}$ is at least as restrictive as the join $\{\text{Chuck}\colon \textit{TopSecret}\} \sqcup_H \{\text{Bob}\colon \textit{Classified}\}$, for $H$ in $\{H_1, H_2\}$. In fact, {Bob: *Classified*, Chuck: *TopSecret*} is equivalent to {Chuck: *TopSecret*} $\sqcup_H$ {Bob: *Classified*}, for $H$ in $\{H_1, H_2\}$.

More generally, for *any* two labels $L_1, L_2 \in \mathcal{L}^{\text{all}}$ and any hierarchy $H$, it is the case that $L_1 \cup L_2$ is equivalent to the join of $L_1$ and $L_2$ in the hierarchy $H$. Thus the set union operation is universal, in that it gives the join for all hierarchies. We formalize this universality property in the following definition.

**Definition 5.2:** Let $\mathcal{L}$ be a class of labels, $\mathcal{H}$ be a class of hierarchies, and $\oplus : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ be a binary operation on $\mathcal{L}$.

- The operation $\oplus$ is a *universal join* for $\mathcal{L}$ with respect to $\mathcal{H}$ if for all $H \in \mathcal{H}$ and $L_1, L_2 \in \mathcal{L}$, it holds that $L_1 \oplus L_2 \equiv_H L_1 \sqcup_H L_2$.
- The operation $\oplus$ is a *universal meet* for $\mathcal{L}$ with respect to $\mathcal{H}$ if for all $H \in \mathcal{H}$ and $L_1, L_2 \in \mathcal{L}$, it holds that $L_1 \oplus L_2 \equiv_H L_1 \sqcap_H L_2$.

∎

**Example 5.3:** As discussed above, the set union operation $\cup$ is a universal join for $\mathcal{L}^{\text{all}}$ with respect to any class of hierarchies $\mathcal{H}$. This can be seen from the proof of Theorem 4.3. ∎

As pointed out in Section 4, the need to express joins of labels is common. The existence of a universal join operation means that our model can express joins of labels even when the run-time hierarchy is unknown.

While it is apparent from Example 5.3 that many instantiations of the model will have a universal join operation, it is not so apparent that universal meet operations exist. The following example shows that they do exist for some instantiations.

**Example 5.4:** Suppose that $\mathcal{H}$ is a class of meet hierarchies where for each hierarchy $H \in \mathcal{H}$, the owner hierarchy $\geq_o^H$ is the equality relation. Moreover, suppose that there is a "universal policy meet" operation $\wedge : P \times P \to P$ such that for each $H \in \mathcal{H}$ and $p_1, p_2 \in P$, the policy $p_1 \wedge p_2$ is equal to the greatest lower bound of $p_1$ and $p_2$ with respect to $\geq_p^H$. Note that the hierarchies of the decentralized label model have a universal policy meet, namely, the set union operation on reader sets, as discussed in Section 3.1.

Recall that $\mathcal{L}^{\text{own-one}}$ denotes the class of labels where each owner appears at most once, defined in Section 3.3.

Define the operation $\oplus : \mathcal{L}^{\text{own-one}} \times \mathcal{L}^{\text{own-one}} \to \mathcal{L}^{\text{own-one}}$ by

$$L_1 \oplus L_2 = \{\mathbf{o}(I) : (\mathbf{p}(I) \wedge \mathbf{p}(J)) \mid I \in L_1, J \in L_2, \mathbf{o}(I) = \mathbf{o}(J)\}$$

It is straightforward to verify that $\oplus$ is a universal meet for $\mathcal{L}^{\text{own-one}}$ with respect to $\mathcal{H}$. ∎

## 5.2. Gaining Partial Knowledge of the Run-time Hierarchy

If the system provides a mechanism for the run-time testing of hierarchies, we may be able to glean partial knowledge about the run-time hierarchy. This additional knowledge can increase the precision of the static analysis.

**Example 5.5:** Suppose $o$ *actsfor* $o'$ is a language mechanism that tests at run time if owner $o$ acts for owner $o'$, that is, $o$ *actsfor* $o'$ is true if and only if $o \geq_o^H o'$ for the run-time hierarchy $H$. Consider the following program, in security typed pseudo-code.

```
1  int{Bob: TopSecret} x := ...;
2  int{Chuck: TopSecret} y;
3  if (Chuck actsfor Bob) {
4          y := x;
5  } else {
6          y := 0;
7  }
```

The assignment at line 4 causes information to flow from the variable $x$ to the variable $y$; in order for this program to be noninterfering, the label of the variable $y$ must be at least as restrictive as the label of the variable $x$.

Assume that $H_1$ and $H_2$ (as defined in Example 5.1) are among the hierarchies that can occur at runtime.

The assignment at line 4 is only executed if the dynamic *actsfor* test evaluates to true, that is, if the owner hierarchy of the run-time hierarchy $H$ includes the relationship (Chuck, Bob). The assignment at line 4 is thus secure, since {Bob: *TopSecret*} $\sqsubseteq_H$ {Chuck: *TopSecret*}, so long as Chuck acts for Bob according to $H$, as in the hierarchy $H_1$.

Note that if we ignored the partial knowledge about the run-time hierarchy revealed by the dynamic *actsfor* test, and assumed that the run-time hierarchy could be $H_2$ even though the assignment at line 4 is executed, then we could not regard the program as secure. This is because Chuck does not act for Bob according to $H_2$, and so {Bob: *TopSecret*} $\not\sqsubseteq_{H_2}$ {Chuck: *TopSecret*}. ∎

Formally, we model the notion of partial knowledge of the run-time hierarchy by the following partial ordering $\le$ on hierarchies.

**Definition 5.6:** The partial ordering $\le$ on hierarchies is defined as follows: for any two hierarchies $H_1$, $H_2$, it holds that $H_1 \le H_2$ if and only if

$$\ge_o^{H_2} \subseteq \ge_o^{H_1} \text{ and } \ge_p^{H_2} \subseteq \ge_p^{H_1} .$$

∎

Intuitively, if $H_1 \le H_2$, then $H_1$ is at least as specific than $H_2$, in that all acts-for relationships present in $H_2$ are present in $H_1$, and similarly for the policy relationships.

## 6. Label Inference

Program analysis can be used to prove that programs have various security properties. This generally requires that items of information manipulated by the program have explicit security labels. The security labels are often specified with program annotations. The need for explicit security labels can impose a significant burden on developers of secure systems, and, in the case of program annotations, clutter program source code. It is thus natural to consider the problem of automatically inferring security labels that are not specified. We call this the label inference problem.

In this section, we study the problem of label inference within our framework. We formalize the label inference problem as a constraint satisfaction problem over a set of constraints of a certain form. We present a general theorem demonstrating that certain parameterizations of the inference problem are polynomial time tractable, and use this general theorem to derive a number of tractability results.

**Definition 6.1:** The Label Inference Problem for a class of labels $\mathcal{L}$ and a class of hierarchies $\mathcal{H}$.

**Input:** A finite set of variables $V$ and a finite set of constraints, each of which has the form

$$a_1 \sqcup \ldots \sqcup a_m \sqsubseteq_{\le H} b_1 \sqcup \ldots \sqcup b_n$$

where $H \in \mathcal{H}$ is a hierarchy, and each $a_i$ and $b_j$ is either a variable from $V$ or a constant label $L \in \mathcal{L}$.

**Question:** Is there a *satisfying interpretation* $\theta : V \to \mathcal{L}$? An interpretation $\theta : V \to \mathcal{L}$ is a satisfying interpretation if for all given constraints

$$a_1 \sqcup \ldots \sqcup a_m \sqsubseteq_{\le H} b_1 \sqcup \ldots \sqcup b_n$$

and all hierarchies $H' \le H$, it holds that

$$\theta(a_1) \sqcup_{H'} \ldots \sqcup_{H'} \theta(a_m) \sqsubseteq_{H'} \theta(b_1) \sqcup_{H'} \ldots \sqcup_{H'} \theta(b_n).$$

(An interpretation $\theta$ is assumed to act as the identity on constant labels $L \in \mathcal{L}$.) ∎

The variables in the constraints represent unknown labels; a program analysis generates a set of constraints such that the program will be considered secure if there a satisfying interpretation. For instance, in Example 5.1, the label $L$ of the program variable $z$ was not specified by the programmer. Suppose that the class of hierarchies $\mathcal{H}$ that may occur at run time is $\{H_1, H_2\}$, where $H_1$ and $H_2$ are as given in Example 5.1. Note that $H_1$ is more specific than $H_2$, that is, $H_1 \le H_2$. The type system of the programming language would require that $L$ satisfy the constraint $\{\text{Chuck}: \textit{TopSecret}\} \sqcup \{\text{Bob}: \textit{Classified}\} \sqsubseteq_{\le H_2} L$ in order for the program to be well-typed and thus secure. The form of constraints we consider are sufficient to represent the constraints used in the type systems of security typed languages such as [23, 6, 19], among others.

Clearly, the Label Inference Problem is contained in the complexity class NP, because an interpretation has a polynomial size representation, and whether or not it is satisfying can be checked in polynomial time. In the case that we have a universal join operation, we can show that Label Inference Problem is polynomial time tractable.

**Theorem 6.2:** *Suppose that $\mathcal{L}$ is a class of labels, and $\mathcal{H}$ is a class of hierarchies. If there is a universal join $\oplus$ for $\mathcal{L}$ with respect to $\mathcal{H}$, then the* Label Inference Problem *for $\mathcal{L}$ and $\mathcal{H}$ is decidable in polynomial time; moreover, there is a polynomial time algorithm that outputs a most restrictive satisfying interpretation, when a satisfying interpretation exists.*

By a *most restrictive satisfying interpretation*, we mean a satisfying interpretation $\theta : V \to \mathcal{L}$ such that for any other satisfying interpretation $\theta'$, it holds that $\theta'(v) \sqsubseteq_{\le H} \theta(v)$ for all $v \in V$ and $H \in \mathcal{H}$.

The proof of Theorem 6.2 is in Appendix A. The proof makes use of the *closure properties framework* for studying the complexity of constraint satisfaction problems [8].

The algorithm is to enforce *generalized arc consistency* (a standard procedure in constraint programming) and then return satisfiable if and only if there is no empty constraint. In the case that there are no empty constraints, the most restrictive satisfying interpretation maps a variable onto the universal join of all of its domain elements.

If the class of labels $\mathcal{L}$ and class of hierarchies $\mathcal{H}$ have a universal meet operation, then an analogous result holds, provided that for each $H \in \mathcal{H}$, the policy hierarchy of $H$ is a distributive lattice. In this case, the algorithm gives a least restrictive satisfying interpretation.

From Theorem 6.2, we can derive the following corollaries.

**Corollary 6.3:** *Let $\mathcal{H}$ be any class of hierarchies. The* Label Inference Problem *for $\mathcal{L}^{\text{all}}$ and $\mathcal{H}$ is decidable in polynomial time.*

**Proof:** As noted in Example 5.3, the class of labels $\mathcal{L}^{\text{all}}$ has a universal join operation with respect to any class of hierarchies $\mathcal{H}$; thus, the result follows from Theorem 6.2. ∎

When there is full knowledge of the run-time hierarchy, or one does not care to reason about different hierarchies that may be in effect at runtime, it is useful to consider a class of hierarchies containing only a single hierarchy. In this case, label inference is tractable for any classes of labels that form a join semilattice.

**Corollary 6.4:** *Let $H$ be any hierarchy and $\mathcal{L}$ be any class of labels such that $\mathcal{L}$ forms a join semilattice under the ordering $\sqsubseteq_H$. The* Label Inference Problem *for $\mathcal{L}$ and $\{H\}$ is decidable in polynomial time.*

**Proof:** If $\mathcal{L}$ forms a join semilattice under the ordering $\sqsubseteq_H$, then the join semilattice operation is a universal join for $\mathcal{L}$ with respect to $\{H\}$; thus, the result follows from Theorem 6.2. ∎

In the case that no owner acts for another, as in Section 3.3, we can use Theorem 6.2 to provide a tractability result.

**Corollary 6.5:** *Let $\mathcal{H}$ be a class of meet hierarchies where for each hierarchy $H \in \mathcal{H}$, the owner hierarchy $\geq_o^H$ is the equality relation, and, there is a "universal policy meet" operation, as in Example 5.4. The* Label Inference Problem *for $\mathcal{L}^{\text{own-one}}$ and $\mathcal{H}$ is decidable in polynomial time.*

## 7. Related Work and Discussion

Our framework for owned policies is inspired by the decentralized label model of Myers and Liskov [16]. The decentralized label model allows owners of information to independently specify information flow restrictions on the data, in the form of sets of permitted readers, as explained in Section 3.1. Our model generalizes the decentralized label model by permitting arbitrary policies for restricting information use, instead of just information flow policies that are reader sets. In addition, our work makes explicit the mechanism for static reasoning about owned policies under partial knowledge of the run-time hierarchy, and clearly distinguishes the notions of universal join/meet versus hierarchy-specific join/meet. The decentralized label model does not make such a distinction clear, and so its label inference algorithm (as presented in [14]), while sound, is not complete and in fact may fail to terminate.

Owner Retained Access Control (ORAC) [11] is a model of owned policies for access control. ORAC uses owned access control lists to approximate the "originator-controlled release" dissemination controls used by the U.S. DoD/Intelligence community. When joining labels, the ORAC model joins the policies owned by the same owner; thus in any ORAC model, a given owner owns at most one policy, as in the class of labels $\mathcal{L}^{\text{own-one}}$, described in Section 3.3. However, in general the ORAC model will not have a universal join operator, making static reasoning about the ORAC model difficult. In [11], ORAC is presented informally only; it is not clear whether the ORAC model can be obtained as an instantiation of our owned policy framework.

While our framework is applicable to several kinds of information use, we see information flow as a primary area of applicability. Following Denning's original work on the lattice model of information flow control [2], there has been much recent work on static information flow control in the form of security typed languages (e.g., [23, 6, 13, 7, 19, 1]). Our work is compatible with most of these efforts, which typically assume some lattice or semilattice structure on labels; as we have shown, there are instantiations of our model satisfying these assumptions.

Many systems need to downgrade information as part of their intended functionality, that is, to relabel information with a more permissive label. For example, a system may be prepared to downgrade some information to allow a user to access it after the user has paid for the information. There has been much recent work focusing on what security guarantees can be made in the presence of downgrading, including *intransitive noninterference* [21, 17, 20], *selective declassification* [18, 16], *robust declassification* [25, 24], *quantitative information flow* (e.g., [12, 10, 3]), and *relative secrecy* [22]. Our framework does not preclude downgrading, and is compatible with most of these efforts, which again assume some structure on labels that instantiations of our framework can satisfy.

Recent work by Zheng and Myers [26] establishes a noninterference result in the presence of dynamic security labels. In their model of dynamic labels, security labels are first class values, and thus, information can be revealed by knowing which security label is used at run time. Our model allows reasoning about dynamic hierarchies through the use of classes of hierarchies, and the constraints described in Section 6. Classes of hierarchies can be used to model dynamic owners and dynamic policies, that is, owners and policies that are first-class values and manipulable at run time. However, our framework does not permit dynamic labels. Extending our model to incorporate dynamic labels will make for interesting future work. This should allow suitable instantiations of our framework to be used to reason about dynamic labels, as in Zheng and Myers' work, and thus provide label inference for languages with dynamic labels.

## 8. Conclusion

We have presented and studied a general framework for owned policies. Owned policies can occur in a variety of contexts, including information flow, access control and software licenses. While placing few restrictions on the structure of owners and policies, the labels of our owned policy model have a lattice structure, and can easily express joins of labels, which arise naturally when information is combined.

Our framework allows static reasoning about owned policies, even when there is only partial knowledge of the hierarchy that will be in effect at run time; this facilitates the static analysis of systems. Static analysis is further facilitated by the framework's amenability to tractable label inference, even with only partial knowledge of the run-time hierarchy. Label inference can also help system development, allowing developers to elide some security label annotations.

## Acknowledgements

We would like to thank Andrew Myers for insights into and discussions about the decentralized label model, and both René Hansen and Andrew for interesting and enlightening discussions about our framework for owned policies.

This work was supported by the Department of the Navy, Office of Naval Research, under ONR Grant N00014-01-1-0968. Any opinions, findings, conclusions, or recommendations contained in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research. This work was also supported by the National Science Foundation under Grant Nos. 0208642 and 0133302.

## A. Proof of Theorem 6.2

**Proof:** We first remark that a constraint

$$a_1 \sqcup \ldots \sqcup a_m \sqsubseteq_{\leq H} b_1 \sqcup \ldots \sqcup b_n$$

can be desugared into the $m$ constraints of the form

$$a_i \sqsubseteq_{\leq H} b_1 \sqcup \ldots \sqcup b_n$$

where $i$ varies from 1 to $m$. It thus suffices to prove the results for constraints of the latter form, which we now proceed to do.

Define $\equiv_{\mathcal{H}}$ to be the equivalence relation such that $L_1 \equiv_{\mathcal{H}} L_2$ if and only if for all $H \in \mathcal{H}$, $L_1 \equiv_H L_2$. Suppose that two interpretations $\theta_1, \theta_2 : V \to \mathcal{L}$ are equivalent up to $\equiv_{\mathcal{H}}$, that is, for all $v \in V$, $\theta_1(v) \equiv_{\mathcal{H}} \theta_2(v)$. It is straightforward to verify that $\theta_1$ satisfies a constraint if and only if $\theta_2$ does. We may thus re-phrase the Label Inference Problem as the problem of deciding if there is an interpretation $\overline{\theta} : V \to \mathcal{L}/\equiv_{\mathcal{H}}$ satisfying every constraint $a \sqsubseteq_{\leq H} b_1 \sqcup \ldots \sqcup b_n$, in the sense that for any $L \in \overline{\theta}(a)$ and $L_1 \in \overline{\theta}(b_1), \ldots, L_n \in \overline{\theta}(b_n)$, it holds that $L \sqsubseteq_{\leq H} L_1 \sqcup_H \ldots \sqcup_H L_n$.

It is straightforward to verify that the universal join operation $\oplus$ is well-defined on elements of $\mathcal{L}/\equiv_{\mathcal{H}}$, that is, if $L_1 \equiv_{\mathcal{H}} L'_1$ and $L_2 \equiv_{\mathcal{H}} L'_2$, then $L_1 \oplus L_2 \equiv_{\mathcal{H}} L'_1 \oplus L'_2$. The operation $\oplus$ thus naturally induces an operation $\overline{\oplus} : (\mathcal{L}/\equiv_{\mathcal{H}}) \times (\mathcal{L}/\equiv_{\mathcal{H}}) \to \mathcal{L}/\equiv_{\mathcal{H}}$. It is straightforward to verify that $\overline{\oplus}$ is a semilattice operation, that is, $\overline{\oplus}$ is associative, commutative, and idempotent. The operation $\overline{\oplus}$ hence induces a partial ordering $\leq_{\overline{\oplus}}$ defined by $S \leq_{\overline{\oplus}} S'$ if and only if $S \overline{\oplus} S' = S'$.

Each constraint $C$ of the described form $a \sqsubseteq_{\leq H} b_1 \sqcup \ldots \sqcup b_n$ has the property that when two interpretations satisfying $C$ are composed via the $\overline{\oplus}$ operation, the resulting interpretation also satisfies $C$. To show this, suppose that the interpretations $\overline{\theta}_1, \overline{\theta}_2 : V \to \mathcal{L}/\equiv_{\mathcal{H}}$ satisfy the constraint

$$a \sqsubseteq_{\leq H} b_1 \sqcup \ldots \sqcup b_n.$$

We claim that the interpretation $\overline{\theta} : V \to \mathcal{L}/\equiv_{\mathcal{H}}$ such that $\overline{\theta}(v) = \overline{\oplus}(\overline{\theta}_1(v), \overline{\theta}_2(v))$ for all $v \in V$, also satisfies the constraint. Indeed, for every hierarchy $H' \leq H$, we have that

$$\overline{\theta}_1(a) \sqsubseteq_{H'} \overline{\theta}_1(b_1) \sqcup_{H'} \ldots \sqcup_{H'} \overline{\theta}_1(b_n)$$

and

$$\overline{\theta}_2(a) \sqsubseteq_{H'} \overline{\theta}_2(b_1) \sqcup_{H'} \ldots \sqcup_{H'} \overline{\theta}_2(b_n),$$

from which it follows that

$$\begin{aligned}&\overline{\theta}_1(a) \sqcup_{H'} \overline{\theta}_2(a) \sqsubseteq_{H'} \\ &\qquad (\overline{\theta}_1(b_1) \sqcup_{H'} \overline{\theta}_2(b_1)) \sqcup_{H'} \\ &\qquad \ldots \sqcup_{H'} (\overline{\theta}_1(b_n) \sqcup_{H'} \overline{\theta}_2(b_n));\end{aligned}$$

we conclude that

$$\begin{aligned}&\overline{\oplus}(\overline{\theta}_1(a), \overline{\theta}_2(a)) \sqsubseteq_{H'} \\ &\qquad \overline{\oplus}(\overline{\theta}_1(b_1), \overline{\theta}_2(b_1)) \sqcup_{H'} \\ &\qquad \ldots \sqcup_{H'} \overline{\oplus}(\overline{\theta}_1(b_n), \overline{\theta}_2(b_n))\end{aligned}$$

or equivalently, that

$$\overline{\theta}(a) \sqsubseteq_{H'} \overline{\theta}(b_1) \sqcup_{H'} \ldots \sqcup_{H'} \overline{\theta}(b_n).$$

(Note that we have implicitly overloaded the $\sqsubseteq_{H'}$ relation and the $\sqcup_{H'}$ operation; the original relation and operation are defined over the set $\mathcal{L}$, but naturally induce a relation and operation defined over the set $\mathcal{L}/\equiv_{\mathcal{H}}$.)

By [8, Theorem 5.13], constraints of the described form are tractable in polynomial time; moreover, the algorithm given in the proof of [8, Theorem 5.13] yields, in the case that the constraints are satisfiable, a satisfying interpretation $\overline{\theta} : V \to \mathcal{L}/\equiv_{\mathcal{H}}$ that is maximal with respect to $\leq_{\overline{\oplus}}$. Any interpretation $\theta : V \to \mathcal{L}$ such that $\theta(v) \in \overline{\theta}(v)$ for all $v \in V$ is a most restrictive interpretation. ∎

## References


[1] A. Banerjee and D. A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *IEEE Computer Security Foundations Workshop (CSFW)*, June 2002.

[2] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[3] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 1–15, June 2002.

[4] GNU general public license. `http://www.gnu.org/copyleft/gpl.html`.

[5] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[6] N. Heintze and J. G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Conference Record of the Twenty-Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 365–377, San Diego, California, January 1998.

[7] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Conference Record of the Twenty-Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 81–92. ACM Press, January 2002.

[8] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

[9] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. In *Proc. 13th ACM Symp. on Operating System Principles (SOSP)*, pages 165–182, October 1991. *Operating System Review*, 253(5).

[10] G. Lowe. Quantifying information flow. In *Proc. 15th IEEE Computer Security Foundations Workshop*, June 2002.

[11] C. J. McCollum, J. R. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC—defining new forms of access control. In *Proc. IEEE Symposium on Security and Privacy*, pages 190–200, 1990.

[12] J. K. Millen. Covert channel capacity. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, 1987.

[13] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Conference Record of the Twenty-Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, TX, January 1999.

[14] A. C. Myers. Mostly-static decentralized information flow control. Technical Report MIT/LCS/TR-783, MIT, Cambridge, MA, January 1999. Ph.D. thesis.

[15] A. C. Myers and B. Liskov. A decentralized model for information flow control. In *Proc. 16th ACM Symp. on Operating System Principles (SOSP)*, pages 129–142, Saint-Malo, France, 1997.

[16] A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *Proc. IEEE Symposium on Security and Privacy*, pages 186–197, Oakland, CA, USA, May 1998.

[17] S. Pinsky. Absorbing covers and intransitive non-interference. In *Proc. IEEE Symposium on Security and Privacy*, pages 102–113, 1995.

[18] F. Pottier and S. Conchon. Information flow inference for free. In *Proceedings of the 2000 ACM SIGPLAN International Conference on Functional Programming*, pages 46–57, 2000.

[19] F. Pottier and V. Simonet. Information flow inference for ML. In *Conference Record of the Twenty-Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 319–330, 2002.

[20] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proc. 12th IEEE Computer Security Foundations Workshop*, 1999.

[21] J. Rushby. Noninterference, transitivity and channel-control security policies. Technical report, SRI, 1992.

[22] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Conference Record of the Twenty-Seventh Annual ACM Symposium on Principles of*

*Programming Languages*, pages 268–276, Boston, MA, January 2000.

[23] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[24] S. Zdancewic. A type system for robust declassification. In *Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics*, Electronic Notes in Theoretical Computer Science, March 2003.

[25] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 15–23, Cape Breton, Nova Scotia, Canada, June 2001.

[26] L. Zheng and A. C. Myers. Dynamic security labels and noninterference. Technical Report 2004–1924, Cornell University Computing and Information Science, 2004.