1976

On the Derivation of Lattice Structured Information Flow Policies

Dorothy E. Denning

Peter J. Denning

G. Scott Graham

Report Number:
76-180

ON THE DERIVATION OF LATTICE STRUCTURED INFORMATION FLOW POLICIES[1]

Dorothy E. Denning[2]
Purdue University

March 1976
CSD TR 180

Recent studies in secure computer systems have shown that lattice structured information flow policies have properties which lead to simple and efficient enforcement mechanisms. This paper outlines a method for transforming nonlattice structured policies into lattices while preserving the validity of all flows.

Key Words and Phrases: protection, security, information flow, security class, lattice

CR Categories: 4.35

---

[1]Work reported herein was supported in part by NSF Grant GJ-43176.

[2]Author's present address: Purdue University, Computer Sciences Dept., W. Lafayette, Ind. 47907.

1. Introduction

Recent research in computer systems security has focused on specifying and enforcing "information flow policies" — i.e., policies regulating information dissemination. It has been shown that lattice structured policies have properties which lead to simple and efficient enforcement mechanisms [1,3,4,8,9]. The program certification mechanism described in [4], for example, verifies that the flow of data through a program does not violate a given lattice structured policy. The lattice properties of the policy are exploited to construct an efficient mechanism that can easily be incorporated into the analysis phase of a compiler.

We have already argued intuitively that lattice structured flow policies follow naturally from the semantics of information flow [3]. The objective of this paper is to show further that lattice structured policies lose no generality. We shall give a construction for transforming an arbitrary flow policy, satisfying minimal conditions, into a lattice while preserving the validity of all flows.

2. Lattice Properties

A lattice is a structure $L = (A, \leq, +, *)$ where $(A, \leq)$ is a partially ordered set (poset) and $+$ and $*$ are, respectively, least upper bound and greatest lower bound operators on A [2,7]. That $(A, \leq)$ is a poset implies that the relation $\leq$ is reflexive, transitive, and anti-symmetric; that is, for all $a,b,c \in A$:

1. $a \leq a$ (reflexive)

2. $a \leq b$ and $b \leq c$ $\Rightarrow$ $a \leq c$ (transitive)

3. $a \leq b$ and $b \leq a$ $\Rightarrow$ $a = b$ (anti-symmetric).

That + is a least upper bound operator on A implies that for each pair of elements a and b in A, there exists a unique element $c \in A$ ($c = a + b$) such that:

1. $a \leq c$ and $b \leq c$, and

2. $a \leq d$ and $b \leq d$ $\Rightarrow$ $c \leq d$ for all $d \in A$.

By extension, corresponding to any nonempty subset $B = \{a_1,\ldots,a_n\}$ of A, there is a unique element $+B = a_1+a_2+\ldots+a_n$ which is the least upper bound for the set. That * is a greatest lower bound operator on A implies that for each pair of elements a and b in A, there exists a unique element e in A ($e = a * b$) such that:

1. $e \leq a$ and $e \leq b$, and

2. $d \leq a$ and $d \leq b$ $\Rightarrow$ $d \leq e$ for all $d \in A$.

By extension, corresponding to any subset $B = \{a_1,\ldots,a_n\}$ of A, there is a unique element $*B = a_1*a_2*\ldots*a_n$ which is the greatest lower bound for the set.

An example of a lattice is derived from the subsets of a given finite set X: $L = (\mathcal{P}(X), \subseteq, \cup, \cap)$, where $\mathcal{P}(X)$ denotes the powerset of X. The partial ordering relation on $\mathcal{P}(X)$ is set inclusion $\subseteq$; the least upper bound operator is set union $\cup$; and the greatest lower bound operator is set intersection $\cap$. Figure 1 illustrates for $X = \{x,y,z\}$. The graphical representation is a standard precedence graph for a partial order, showing only the non-reflexive immediate relations. Subset lattices play a key role in the transformation of flow policies into lattices.

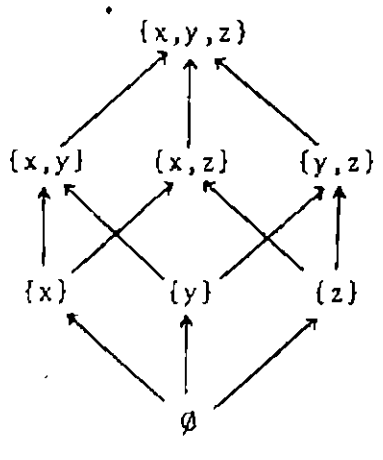


Figure 1. Lattice of subsets of X = {x,y,z}.

3. Information Flow Policies

An information flow policy P is defined by P = (X, →), where X is a set of "security classes" and → is a flow relation on X. P is a lattice structured policy if it is a poset and there exist least upper and greatest lower bound operators, denoted ⊕ and ⊗ respectively, on X.

For security classes x and y, we write x → y if and only if information in class x is permitted to flow into class y. Objects such as files, variables, and users are associated with security classes; flows between two security classes x and y result when information derived from an object with security class x is stored in another object with security class y. A complete description of flow policies and an overview of their enforcement mechanisms is given in [3].

An alternative specification for information flow policies was studied by Jones and Lipton [6]. They defined a policy in terms of allowable flows from the input parameters I to the output parameters O of a program. This type of policy can be expressed in our terms as a

lattice structured flow policy $(\mathcal{P}(I), \subseteq, \cup, \cap)$ over security classes $\mathcal{P}(I)$: each input parameter i is associated with the security class {i}, and each output parameter o is associated with the security class {i | i may flow into o}.

4. Derivation of Lattice

The objective is to transform a given flow policy into a lattice while preserving the validity of all flows. To make this precise, let $P = (X, \rightarrow)$ and $P' = (X', \rightarrow')$ be flow policies and let f be a mapping from X to X'. Then f is flow preserving between P and P' if and only if for all $x,y \in X$, $x \rightarrow y \iff f(x) \rightarrow' f(y)$. This says that the flows permitted by policy P between pairs of security classes $x,y \in X$ are the same as those permitted by policy P' between security classes $f(x),f(y) \in X'$. Let $\underline{o}$ denote the security class in X associated with object o. Then if $\underline{o}$ is replaced by the security class $f(\underline{o})$ in X' for all objects o, policy P' will permit exactly the same flows to occur among objects as policy P.

Our method for transforming a policy $P = (X, \rightarrow)$ into a lattice requires that P satisfy two conditions:

1. X must be finite, and
2. $\rightarrow$ must be reflexive and transitive.

The first condition is required to guarantee the transformation halts; the second is required to construct a flow preserving mapping. We are aware of no policies which violate either of these conditions. There are, however, policies which satisfy these conditions but not necessarily the stronger conditions required for a lattice (c.f. [5]).

Define a mapping f from X to $\mathcal{P}(X)$ by $f(x) = \{y \mid y \in X \text{ and } y \rightarrow x\}$ for all $x \in X$. This maps each security class x into the set consisting of all of the security classes permitted to flow into x. Let $A = \{f(x) \mid x \in X\}$ and consider a policy $Q = (A, \subseteq)$ that orders members of A by set inclusion. We make the following observation:

Lemma 1. f is flow preserving between P and Q.

Proof. Consider any $x,y \in X$. We must show that $x \rightarrow y \iff f(x) \subseteq f(y)$. Suppose first that $x \rightarrow y$, and let $z \in f(x)$. Since $z \rightarrow x \rightarrow y$ implies $z \rightarrow y$ by transitivity of $\rightarrow$, $z \in f(y)$, and thus $f(x) \subseteq f(y)$. Suppose next that $f(x) \subseteq f(y)$. Since $\rightarrow$ is reflexive, $x \in f(x)$; but then $x \in f(y)$, which implies $x \rightarrow y$.

The above policy Q eliminates redundant security classes from P to achieve a partial ordering of the classes. For example, if both $x \rightarrow y$ and $y \rightarrow x$ holds for security classes x and y, one is redundant since anything in one can flow to the other. Since $f(x) \subseteq f(y)$ and $f(y) \subseteq f(x)$ imply $f(x) = f(y)$, this redundancy is removed in policy Q.

Although $Q = (A, \subseteq)$ is a poset, it may not satisfy the least upper bound and greatest lower bound lattice properties. To obtain a policy $R = (B, \subseteq)$ which does satisfy these properties, we first set $B = A \cup \emptyset \cup X$, where the empty set $\emptyset$ gives a lower bound on B, and the set X gives an upper bound on B. We then add to B additional sets of $\mathcal{P}(X)$ until each pair of elements of B has a least upper bound in B. To determine if a pair of elements $a,b \in B$ has a least

upper bound in B, we identify the set of successors common to a and b, $S(a,b) = \{d \mid d \in B \text{ and } a \cup b \subseteq d\}$. Clearly S(a,b) is nonempty since $X \in S(a,b)$. Let $c = \cap S(a,b)$. If $c \in S(a,b)$, then a and b have c as their least upper bound in B; otherwise they have none, so we add c to B.

This expansion of B is guaranteed to terminate since there are at most a finite number of sets in $\mathcal{P}(X)$ that can be added to B, and set union is a least upper bound operator on $\mathcal{P}(X)$. Let $\oplus$ denote the least upper bound operator on B, and let $R = (B, \subseteq, \oplus)$ denote the resulting structure. The following theorem proves that the derivation is flow preserving and yields a lattice structured policy whose flow relation is set inclusion.

Theorem. $R = (B, \subseteq, \oplus)$ is a lattice for which f is flow preserving between P and R.

Proof. To prove that R is a lattice, we need only show that there exists a greatest lower bound operator on B. For any pair of sets $a,b \in B$, define $T(a,b) = \{c \mid c \in B \text{ and } c \subseteq a \cap b\}$. T(a,b) is the set of predecessors common to a and b. Clearly T(a,b) is nonempty since the empty set $\emptyset$ is in T(a,b). Define $a \otimes b$ to be the least upper bound of the predecessors: $a \otimes b = \oplus T(a,b)$. Then $a \otimes b \in B$ by construction of B, and $a \otimes b \subseteq a$ and $a \otimes b \subseteq b$. Also, if $c \subseteq a$ and $c \subseteq b$, then $c \subseteq a \otimes b$. Therefore, $a \otimes b$ is a greatest lower bound operator on B.

To prove that f is flow preserving between P and R, it is sufficient to observe that for any pair of security classes x and

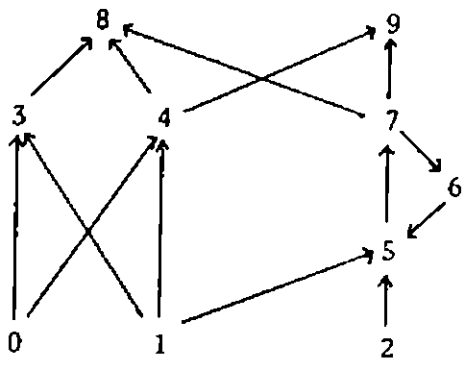
y in X, the sets f(x) and f(y) are ordered by set inclusion in R as in Q. Therefore, the results of Lemma 1 hold for both Q and R.

5. Example

Figures 2 (a)-(d) illustrate the derivation of a lattice from an initial policy P = (X,→). The security classes in X are numbered 0,1,...,9. Note that the flow preserving mapping f takes the security classes 5, 6, and 7, which form a cycle, into the single set {1,2,5,6,7}. The expansion of the set $\{f(x) \mid x \in X\}$ into a lattice requires the addition of only four sets: ∅, {0,1}, {0,1,2,4,5,6,7} and X. These new sets are distinguished in the graph of the final lattice with asterisks.

6. Conclusions

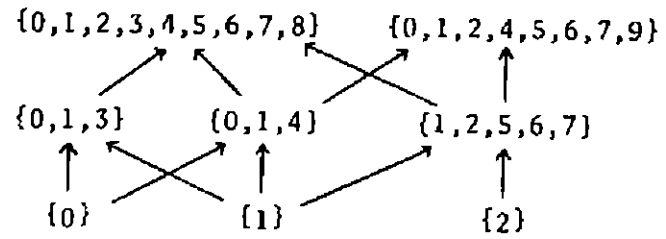
We have supported our assertion that lattice structured information flow policies lose no generality. This was proved by showing that an arbitrary finite, reflexive, and transitive flow policy can be transformed into a lattice while preserving the validity of all flows. We have omitted the details of this transformation because 1) our primary objective was to demonstrate the existence of a flow preserving lattice transformation; and 2) we believe that for most systems, the transformation would be required only once and could be done by inspection. The design of an efficient transformation which derives lattices having a minimal number of sets would be of interest in systems where the policy is subject to frequent change.



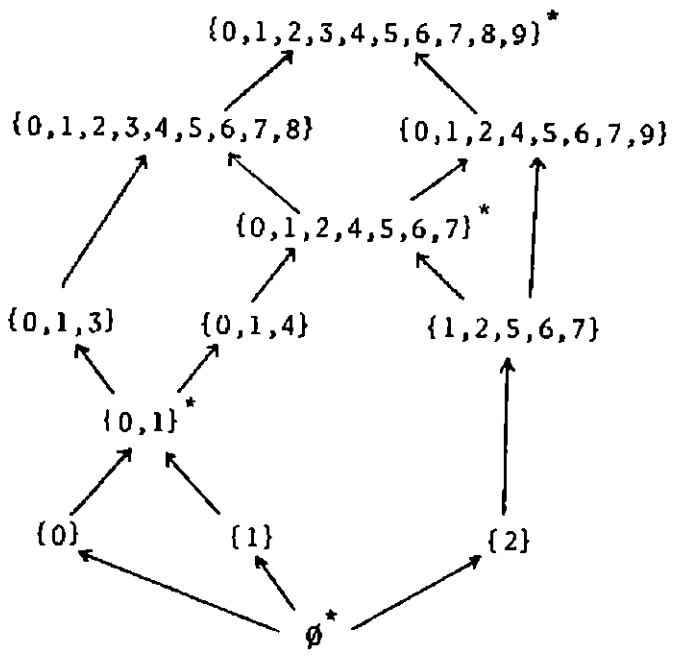
a. Initial policy P = (X, →).

x	f(x)
0	{0}
1	{1}
2	{2}
3	{0,1,3}
4	{0,1,4}
5	{1,2,5,6,7}
6	{1,2,5,6,7}
7	{1,2,5,6,7}
8	{0,1,2,3,4,5,6,7,8}
9	{0,1,2,4,5,6,7,9}

b. Flow preserving mapping.



c. Intermediate policy Q = (A, ⊆).



d. Final lattice structured policy R = (B, ⊆).

Figure 2. Example of a Lattice Derivation.

Acknowledgments.

I am grateful to Peter J. Denning for many helpful suggestions.

References.

1. Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations," ESD-TR-73-278, Vol. I-III, The MITRE Corp , Bedford, Mass.

2. Birkhoff, G. Lattice Theory,Amer. Math. Soc. Col. Pub., XXV, 3rd. ed., 1967.

3. Denning. D. E., "A Lattice Model of Secure Information Flow," Comm. ACM, May 1976.

4. Denning, D. E. and Denning, P. J., "Certification of Programs for Secure Information Flow," Comm. ACM, (to appear).

5. Fenton, J. S.. "Information Protection Systems," Ph.D. Dissertation, Univ. Of Cambridge, England, 1973.

6. Jones. A. K. and Lipton, R. J., "The Enforcement of Security Policies for Computation," Proc. Fifth Symposium on Operating Systems Principles, Nov. 1975.

7. Stone, H. S., Discrete Math. Structures and Their Applications, Science Res. Assoc., 1973.

8. Walter, K. G. et al., "Structured Specification of a Security Kernel," Proc. Int'l Conf. on Reliable Software, SIGPLAN Notices. 10, 6, June 1975, 285-293.

9. Weissman, C., "Security Controls in the ADEPT-50 Time-Sharing System," Proc. AFIPS 1969 FJCC, 35, 119-133.