A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University - 1976

Presented by Adam Fuchs
1969 - Arpanet begins, First ATM  
1971 - Micro Processor, Floppy Disk  
1972 - Pocket Calculator  
1973 - Micro Computer Commercially Available  
1974 - First use of "Internet"  
1975 - This paper written  
1976 - CRAY-1, Apple I  
1977 - PC MODEM, Apple II  
1978 - TCP/IPv4
Secure Information Flow

"...no unauthorized flow of information is possible."

Good for securing:
- Classified computer systems
- Multi-user systems
- Inter-process communication (collaborating users)
- Inadvertent information leaks
Flow Model

\[ FM = \langle N, P, SC, \odot, \rightarrow \rangle \]

\( N = \{a, b, \ldots\} \)
Storage Objects (Inputs, Outputs, Program Variables)

\( P = \{p, q, \ldots\} \)
Processes

\( SC = \{A, B, \ldots\} \)
Security Classes (Unclassified, Secret, etc.)

\( \odot \) : Class Combining Operator
\[(SC \times SC) \rightarrow SC\]

\( \rightarrow \) : Flow Relation
e.g. Public \( \rightarrow \) Private, but not Private \( \rightarrow \) Public
Object a has security class binding a

Static Binding: a remains constant
  Particularly useful for input/output channels

Dynamic Binding: a changes depending on its contents
  Particularly useful for program variables
Class-Combining Operator ☻

Private ☻ Public = Public
Unclassified ☻ Secret = Secret

f(a,b,c) has security class a ☻ b ☻ c

SC is closed under ☻
Flow Relation → and Security Requirements

Public → Private is defined
Private → Public is not

"A flow model $FM$ is secure if and only if execution of a sequence of operations cannot give rise to a flow that violates the relation '→'.'"

If we have a statement $d = f(a,b,c)$, then $a \circ b \circ c \rightarrow d$ must hold.

Transitivity implies that local security implies overall security.
Lattice Structure

A lattice structure has:
- A partially ordered set of elements $S$
- A least upper bound defined on $S \times S$
- A greatest lower bound defined on $S \times S$
- A top element that is an upper bound for all elements in $S$
- A bottom element that is a lower bound for all elements in $S$

Fig. 1. Linear ordered lattice.

$SC = \{A_1, \ldots, A_n\}$

$A_i \rightarrow A_j$ iff $i \leq j$

$A_i \oplus A_j \equiv A_{\max(i,j)}$

$A_i \otimes A_j \equiv A_{\min(i,j)}$

$L = A_1; \quad H = A_n$

Description | Representation
--- | ---
$A_n$ | $A_{n-1}$
$A_{n-1}$ | $A_{n-2}$
$A_{n-2}$ | $A_{n-3}$
$\vdots$ | $\vdots$

Fig. 2. Lattice of subsets of $X = \{x, y, z\}$.

$SC = \text{powerset } (X)$

$A \rightarrow B$ iff $A \subseteq B$

$A \oplus B \equiv A \cup B$

$A \otimes B \equiv A \cap B$

$L = \emptyset; \quad H = X$

Description

Representations
Lattice Applied to Information Flow

Four assumptions show that \( \langle SC, \circ, \rightarrow \rangle \) forms a lattice:
1. \( \langle SC, \rightarrow \rangle \) is a partially ordered set.
2. SC is finite.
3. SC has a lower bound \( L \) such that \( L \rightarrow A \) for all \( A \) in SC.
4. \( \circ \) is the least upper bound operator on SC.

Properties of \( \rightarrow \):
1. Reflexive: \( A \rightarrow A \)
2. Transitive: \( A \rightarrow B \) and \( B \rightarrow C \) implies \( A \rightarrow C \)
3. Antisymmetric: \( A \rightarrow B \) and \( B \rightarrow A \) implies \( A = B \)
Explicit vs. Implicit Flow

Explicit: $a := b$, so $b \rightarrow a$

Implicit: $\textbf{if } a = 0 \textbf{ then } b := c$, so $a \rightarrow b$
Abstract Program Representation

An abstract program (or statement) $S$ is defined recursively by:
1. $S$ is an elementary statement; e.g. assignment or I/O
2. There exist $S_1$ and $S_2$ such that $S = S_1; S_2$ (Sequence)
3. There exist $S_1,...,S_m$ and an $m$-valued variable $c$ such that $S = c : S_1,...,S_m$ (Conditional)

while $c$ do $S$ and if $c$ then $S$ are both represented by $c : S$
1. Explicit flow caused by an elementary statement must be secure.
2. Each part of a sequence statement must be secure.
3. Each part of a conditional statement must be secure, and any implicit flow in a conditional statement must be secure.

Consider the statement $S = c:S_1,...,S_m$. Let $b_1,...,b_n$ be the objects that receive explicit flow in $S_1$ through $S_m$. The implicit flow in $S$ is $c \rightarrow b_i$, $1 \leq i \leq n$. 
Each process $p$ has a static binding $p$.
$p$ can read from object $a$ IFF $a \rightarrow p$.
$p$ can write to object $b$ IFF $p \rightarrow b$.
Transitivity guarantees that $a \rightarrow b$, so implicit flows are also secure.
Enforcement happens at run-time.

This mechanism adds restrictions to the model:
Public $\rightarrow$ Public and Private $\rightarrow$ Private data transfers must use separate processes.
Extension to the Case and MITRE systems.
- Data is marked with its security class.
- For program $p$, $p$ changes according to the control flow.
- Upon entering statements $S = c:S_1,...,S_m$, $p$ is pushed onto the stack and replaced with $p \odot c$.
- Explicit flow statements $a := ...$ also require that $p \rightarrow a$.

This is less restrictive than Case and MITRE Systems.
Static Binding - Certification Mechanism

- Information flow security is checked at compile time.
- Each statement gets the lower bound of the security classes of objects receiving explicit flow.
  - For $S = (a := b)$, $S = a$, and $b \rightarrow a$ is checked.
  - For $S = S_1; S_2$, $S = S_1 \circ S_2$
  - For $S = c : S_1, ..., S_m$, $S = S_1 \circ ... \circ S_m$, and $c \rightarrow S$ is checked.

This constitutes early static analysis, and is sound.
Dynamic Binding Mechanisms

- Dynamic binding allows security classes to change throughout the execution of a program.
- Requires some static bindings to relate to the real world.
- Vulnerable to leaking data through covert channels.
Dynamic Binding - Dynamic Data Mark Machine

Instead of checking that a dynamically bound object can receive flow, update the security class of that object:

- The check of $a \circ b \circ c \rightarrow d$ becomes $d := a \circ b \circ c$

Statically bound objects are still checked (must be at runtime).

Flaw: Implicit flows are not checked in the absence of explicit flows.
Dynamic Data Mark Machine Example

\[ b := c := \text{false}; \]
\[ \text{if} \sim a \text{ then } c := \text{true}; \]
\[ \text{if} \sim c \text{ then } b := \text{true} \]

Assuming that the constants \textit{true} and \textit{false} are in the least class \( L \), execution of this program by a process \( p \) proceeds as follows:

\[ b := c := \text{false}; \quad b := c := L; \]
\[ p := a; \quad \text{if} \sim a \text{ then } \{ c := \text{true}; \quad c := L \oplus p \}; \quad p := L; \]
\[ p := c; \quad \text{if} \sim c \text{ then } \{ b := \text{true}; \quad b := L \oplus p \}; \quad p := L \]

Since \( p = L \oplus p \), this simplifies to:

\[ b := c := \text{false}; \quad b := c := L; \]
\[ \text{if} \sim a \text{ then } \{ c := \text{true}; \quad c := a \}; \]
\[ \text{if} \sim c \text{ then } \{ b := \text{true}; \quad b := c \} \]
Dynamic Binding - Nondecreasing Class Mechanisms

Security classes change monotonically:
- $b := a$ results in $b := b \odot a$, even though $b$ is overwritten
- $p$ also does not decrease.

Still suffers from implicit flow being unchecked in the absence of explicit flow in multi-process collaborations.

Other minor flaws may exist in some implementations...
Discussion: Flaws in the Model

Dissemination and declassification are not modeled
- Voting machine can release a sum of votes
- Actions based on classified data happen at observable security classes

Application of function to data can raise security class, so ☺ may be function specific