A Tale of Two “Flow” Theorems

George W. Dinolt*
Computer Science Dept.
Naval Postgraduate School

*mailto:gwdinolt@nps.edu
Outline

- Information Flows
- The Bad — Harrison-Ruzzo-Ullman
- The Better — Denning Lattice
- The Consequences
Our Definition of Information Flow
Some Terms

**Information:** A sequence of bits that has an interpretation (usually by some program).†

**Container:** Something that holds “information”.

**Information Flow:** A movement of a sequence of bits or a function of those bits from one container (the source) to another (the sink). ‡

†Hopefully only one interpretation is possible, but ...?
‡The source may be the same as the sink
Example Flow - Graph
A “1” for flow from “row position” to “column position”, blank or “zero” otherwise

\[
\begin{array}{cccc}
& C_1 & C_2 & C_3 & C_4 \\
C_1 & & 1 & 1 \\
C_2 & & & 1 \\
C_3 & 1 & 1 & \\
C_4 & 1 & \\
\end{array}
\]
**Object:** Something that contains information, a Container

**Subject:** Containers that may read information from an Object, processes (interprets) the information and writes it to another Object. Note that Subjects may take on the role of Object – Information may flow directly between Subjects.
Access Rights

Normally a Subject may:

**Read:** from an Object — Information flows from the Object to the Subject or

**Write:** to an Object — Information flows from the Object to the Subject
CS View – Graph

\[
\begin{align*}
Ob_1 &\rightarrow Sb_1 \xrightarrow{W} Ob_2 \\
Sb_1 &\rightarrow Ob_1 \xrightarrow{R} Sb_2 \\
Ob_2 &\rightarrow Sb_2 \xrightarrow{W} Ob_1 \\
Sb_2 &\rightarrow Ob_2 \xrightarrow{R} Sb_1
\end{align*}
\]
CS View – Access Matrix

The Computer Science/Computer Security View:

<table>
<thead>
<tr>
<th></th>
<th>$Ob_1$</th>
<th>$Ob_2$</th>
<th>$Sb_1$</th>
<th>$Sb_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ob_1$</td>
<td></td>
<td>RW</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>$Ob_2$</td>
<td></td>
<td>W</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>$Sb_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Sb_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Modifying the Flows — Access Rights

According to system “rules”, Subjects may modify the Flow with the following Actions:

**Create:** Create a new Subject or Object

**Delete:** Destroy a Subject or Object (erasing its contents)

**Add Access:** Add a flow between a Subject and an Object

**Delete Access:** Delete a flow between a Subject and an Object

**Modify System Structure:** Allow creation and deletion of Subject and Objects

**Change Permissions to Modify Flows:** A *meta-action*, give or take the ability to modify the system Flows to another Subject.
The Fundamental Problem

Given:

- An initial set of Subjects and Objects,
- An initial set of Flows,
- A set of “rules” for adding/deleting Subjects and Objects and (creating or deleting rows and columns)
- A set of “rules” for changing the flows (entering/deleting accesses)

Predict System Information Flow.
The Results

**Theorem 1** In general, if one allows for the creation of new Subjects and the addition of Flows, predicting Information Flows is equivalent to solving the “Halting Problem”.

**Theorem 2** Of one wants to be able to create Subjects and add Flows, then the underlying information structure must conform to a Lattice if one wants to be able to Predict Information Flows.
The HRU Protection Model

Given:

- Subjects, $S$, Objects, $O$, Access Rights, $\rho$
- System State defined by an Access Matrix $A$ where
  - Rows are the Subjects
  - Columns are the Objects
  - Entries are “sets of access rights”
- System Transitions - sequences of $create$, $destroy$, $enter$, $delete$ commands
Commands in the Protection Model

Let $PS = \langle S, O, \rho, C_i \rangle$ be a Protection System where the $C_i$ are the commands of the protection system indexed by $i$. Each command is one of form:

$$Command \ C_i(\langle s \rangle, \langle o \rangle)$$

if

$$r_1 \in A[s_k, o_m] \land$$
$$r_2 \in A[s_j, o_l] \land$$
$$\ldots$$

then

$$enter(r_t, A[s_x, o_y]) \land$$
$$delete(r_x, A[s_u, o_v]) \land$$
$$\ldots$$

The notation $\langle x \rangle$ denotes a sequence of elements from the set $x$. $x_k$ is the “k-th” element of the sequence.
Predicting Access

Suppose $r \in \rho$ ($r$ is some access right)

$Q$ is an *unauthorized state* if there is some command that would end with $r$ in some part of $A$ that originally did not contain it.

$Q_0$ is said to be *safe* for $r$ if it cannot derive a state $Q$ in which $r$ could be leaked.

I.E. Can we predict whether a flow will occur in the “future”? 
Undecidability of the Protection Problem

Theorem 1 It is undecidable whether a given state of a protections system is safe for a given generic right.

Outline of Proof: Show that a turing machine can be “modelled” by a protection system with the “states” of the machine mapped to the “rights” of the protection system. We let \( q_f \) map to the safety property. If we could determine whether a system was safe or not, we could equivalently solve the “halting problem”.

Let \( T = \langle K, \Gamma, \{L, R\}, \delta, q_f \rangle \) be a Turing Machine.

Where

\( K \) are the states

\( \Gamma \) is the alphabet or symbols

\( q_f \) is the final or potential termination symbol

\( \delta \) is the transition function

\( L, R \) indicate the Left or Right movement on the head on the “tape”

\[ \delta : K \times \Gamma \rightarrow K \times \Gamma \times \{L, R\} \]
To start with, there is a “tape” that has some symbols “on it”, the system. There is a “head” or tape reader, that starts at tape position “0”. The head “reads” the symbol from its current position on the tape, and based on the symbol and current state, rewrites the symbol at that position, changes the state and moves the head either left or right.

The “Halting Problem is”:

Given an initial “tape”, a \( \delta \) function and a “final state” \( q_f \), will the system ever reach \( q_f \).
Modeling a Protection System

We need to show how the elements of a Turing machine map to elements of a “Protection System”

- The Access Rights: \( \rho = \Gamma \cup K \cup \{Own, End\} \) where \( \Gamma \) are the tape symbols and \( K \) the states of the Turing Machine. The symbols Own and End are new Access Rights that we have created for our ends.

- The Subjects: \( S = \) tape cells. We will name the subjects by the position of the tape cell.

- The Objects: \( O = S \), every subject is also an object. There are no additional objects.

- The Access Matrix: \( \subset 2^{S \times O \times \rho} \) (the set of all subsets of \( S \times O \times \rho \))
Initial State

\[ A = \{(s_0, s_0, q), (s_0, s_0, End), (s_0, s_0, X_0), (s_1, s_1, X_1), (s_2, s_2, X_2), \ldots \} \]

where the \( X_i \) are the symbols on the Turing Machine tape at the "beginning of time" and \( q \) is the initial state.
## View of Initial State

<table>
<thead>
<tr>
<th>tape pos</th>
<th>( s_0 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( \ldots )</th>
<th>( s_n )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( {q_0, X_0, End} )</td>
<td></td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( s_1 )</td>
<td></td>
<td>( {X_1} )</td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( s_2 )</td>
<td></td>
<td></td>
<td>( {X_2} )</td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( s_n )</td>
<td></td>
<td></td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td></td>
</tr>
</tbody>
</table>
The $\delta$ function

- Moves of $T$, $\delta(q, X)$ are represented by commands of the Protections System. For each state $q \in K$ and for each $X \in \Gamma$ there is a command $C_{qX}$ that indicates how the protections system will change when the system is in state $q$ and is “reading” the symbol $X$ from the tape.

- The Turing Machine is at position $j$ on the tape at the beginning of the move. It will be at either $j + 1$ or $j - 1$ at the end of the move.

- There are two kinds of commands, “Left” moves and “Right” moves.
Left Moves

Assume the tape head is at position $j$. 
$\delta(q, X) = (p, Y, L)$ is represented by the commands:

\[
C_{qX}(s_{j-1}, s_j) = \\
\text{if} \\
Own \in A[s_{j-1}, s_j] \land \\
q \in A[s_j, s_j] \land \\
X \in A[s_j, s_j] \\
\text{then} \\
delete q \text{ from } A[s_j, s_j] \\
delete X \text{ from } A[s_j, s_j] \\
\text{enter } Y \text{ into } A[s_j, s_j] \\
\text{enter } p \text{ into } A[s_{j-1}, s_{j-1}]
\]
**Example Left Move**

<table>
<thead>
<tr>
<th>tape pos</th>
<th>$s_0$</th>
<th>$\cdots$</th>
<th>$s_{j-1}$</th>
<th>$\downarrow$</th>
<th>$s_j$</th>
<th>$\cdots$</th>
<th>$s_k$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>${X_0}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>${Own}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$s_{j-1}$</td>
<td>$\cdots$</td>
<td>${Z}$</td>
<td>${Own}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\rightarrow s_j$</td>
<td>$\cdots$</td>
<td>${X,q}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>

**Before:**

<table>
<thead>
<tr>
<th>tape pos</th>
<th>$s_0$</th>
<th>$\cdots$</th>
<th>$s_{j-1}$</th>
<th>$\downarrow$</th>
<th>$s_j$</th>
<th>$\cdots$</th>
<th>$s_k$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${X_0}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>${Own}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\rightarrow s_{j-1}$</td>
<td>$\cdots$</td>
<td>${Z,p}$</td>
<td>${Own}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$s_j$</td>
<td>$\cdots$</td>
<td>${Y}$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>

**After:**
Every Turing Machine can be modeled as a safety problem of a Protections System.

In general, if one could predict whether the safety problem was solvable,

One could solve the “Halting Problem”.

So in general, trying to predict flows using the normal discretionary access controls is not possible.
The model is: $FM = < N, P, SC, \oplus, \rightarrow >$

- $N = \{a, b, c, \ldots\}$ – storage objects
- $P = \{p, q, r, \ldots\}$ – Processes
- $SC = \{A, B, \ldots\}$ – The security classes (labels)
- $lb : N \cup P \rightarrow SC$ a labeling function
- $\oplus : SC \times SC \rightarrow SC$ – class combining function, associative and commutative, extend to many arguments
- $\rightarrow : SC \times SC$ a relation on $SC$, $A \rightarrow B$ implies information can flow from $A$ to $B$. Flows result from operations that cause information to move from $A$ to $B$

§Taken From Denning’s A Lattice Model of Secure Information Flow, Communications of the ACM, Vol 19, #5, May, 1976
The actions of the processes are made up of functions

\[ f : N^n \to N \]

The functions take contents of storage objects as inputs and as part of the “return” cause the values in some storage object to change. This models the details of the “information flow”.

A process will be a sequence of such functions.

We assume that there is only one process.

Footnote: On a single processor with one cpu, there really is only a single sequence of actions. On a multi-cpu system, life is much more interesting.
Security Model

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

If $f(a_1, a_2, \ldots, a_n)$ is a function on objects with results an object $b$, then

$$lb(a_1) \oplus lb(a_2) \oplus \ldots \oplus lb(a_n) \rightarrow lb(b)$$
Assumptions about Flows

We argue that these are natural assumptions about security labels and their combination.

- $\rightarrow$ is Reflexive - $lb(a) \rightarrow lb(a)$

- $\rightarrow$ is transitive
  \[(lb(a) \rightarrow lb(b) \text{ and } lb(b) \rightarrow lb(c)) \Rightarrow lb(a) \rightarrow lb(c)\]

- $lb(a) \rightarrow ((lb(a) \oplus lb(b)) \text{ and } lb(b) \rightarrow (lb(a) \oplus lb(b)))$

- $(lb(a) \rightarrow lb(b)) \Rightarrow (lb(a) \oplus lb(b)) = lb(b)$ \[\|\]

- The set of labels is finite.

\[\|\text{This assumption is not explicitly stated in the paper, but turns out to be the result of one part of the discussion, which I will point out below}\]
POSET Representation of a Lattice

Let $S$ be a set and $\leq$ be a relation on $S$ that satisfies

- $\forall s \in S : \ s \leq s$ (reflexive)
- $\forall s, t \in S : \ (s \leq t) \land (t \leq s) \Rightarrow s = t$ (antisymmetric)
- $\forall s, t, u \in S : \ (s \leq t) \land (t \leq u) \Rightarrow s \leq u$ (transitivity)
- $\forall s, t \in S : \ \exists u \in S, u = \text{lub}(s, t)$ (least upper bound)**
- $\forall s, t \in S : \ \exists w \in S, w = \text{glb}(s, t)$ (greatest lower bound)

**$u$ is a least upper bound of $s$ and $t$ if $s \leq u$ and $t \leq u$ and if $s \leq v$ and $t \leq v$ then $u \leq v$ ($u$ is the smallest upper bound of $s$ and $t$)
Under the above assumptions, the set of “distinguished” †† labels form a lattice, where the “comparison” operator is “flows”, the “→” relation, and where we will define the \( glb \) and \( lub \) functions using “→” and “⊕”.

††We will formally define th is below
Equivalent Security Classes

Two elements \( A, B \in SC \) are equivalent \( \iff \) (if and only if) \( A \rightarrow B \) and \( B \rightarrow A \).

We denote by \( \bar{A} = \{ B : A \rightarrow B \text{ and } B \rightarrow A \} \) the set of all the elements equivalent to \( A \).

It is easy to show that if \( \bar{A} \cap \bar{B} \neq \emptyset \) then \( \bar{A} = \bar{B} \).

We call \( \bar{A} \) the equivalence set of \( A \) in \( SC \).

We will show that the collection of distinct equivalent sets in \( SC \) form a lattice. From here on in, we will assume that we are dealing with the equivalence classes, i.e. that if \( A \rightarrow B \) and \( B \rightarrow A \) then \( A = B \).

\( \dagger \dagger \) This is the definition of “distinguished”
We show that \( \to \) is reflexive, antisymmetric and transitive:

We have reflexivity and transitivity by the definition of \( \to \).

We have anti-symmetry by the previous discussions.

We define \( \text{lub}(A, B) = A \oplus B \) and we show that \( A \oplus B \) has the properties of an upper bound.

From our assumptions, we have \( A \to A \oplus B \) and \( B \to A \oplus B \) so \( A \oplus B \) is an upper bound of \( A \) and \( B \).
Suppose $C$ is another upper bound of $A$ and $B$, that is that $A \rightarrow C$ and $B \rightarrow C$. Then, using our assumptions: $C = A \oplus C$, $C = B \oplus C$ and $(A \oplus B) \rightarrow (A \oplus B) \oplus C$.

But

\[
A \oplus B \rightarrow A \oplus (B \oplus C) \\
\rightarrow A \oplus C \\
\rightarrow C
\]

using the substitutions above.

So $(A \oplus B) \rightarrow C$ for an arbitrary upper bound $C$ and hence $\text{lub}(A, B) = A \oplus B$. 

If $SC$ has an element 0 such that

$$\forall A \in SC, \ 0 \rightarrow A, \ \text{and} \ 0 \oplus A = A \oplus 0 = A$$

then we are done.

If no such element exists, we can just add a new element to $SC$ that satisfies that property.

We do all of our reasoning on this new set.
The \textit{glb} function

We can easily extend the \textit{lub} function to sets.

\[
lub(A, B, C) = lub(A, lub(B, C)) = A \oplus B \oplus C
\]

Suppose \(A, B \in SC\). Consider the set

\[
\mathcal{X} = \{C : C \rightarrow A \text{ and } C \rightarrow B\}
\]

We know \(\mathcal{X} \neq \emptyset\) since \(0 \in \mathcal{X}\). We define \(glb(A, B) = lub(\mathcal{X})\).
The *glb* function continued

We need to show that \( glb(A, B) \to A, \ glb(A, B) \to B \) and \( (Z \to A \land Z \to B) \Rightarrow Z \to glb(A, B) \).

Suppose \( A \neq glb(A, B) \) and \( A \to glb(A, B) \). But \( \forall x \in X, \ x \to A \), and so \( A \) is an upper bound for \( X \), so \( lub(X) \to A \) which contradicts the definition of \( lub \). We can argue similarly for \( B \).

We need to show that there isn't any “bigger” element that could be a *glb*. Let \( (Z \to A) \land (Z \to B) \land (glb(A, B) \to Z) \). Clearly \( Z \in X \), hence \( Z \to lub(X) = glb(A, B) \) which is a contradiction unless \( Z = glb(A, B) \).
SC is a Lattice

$SC$, $lub$, $glb$ ($SC$, $\oplus$, $\rightarrow$) forms a lattice when the “duplicate” classifications are removed and the 0 element is added.
Why this is Important

If we are interested in understanding and controlling information flow in a system, then, under some very minimal assumptions, the labels associated with the information flow must form a lattice.

Even if the Labels are not “explicit”, there are implicit labels, for example, each container can have its own label.

We have seen that attempting to predict access using discretionary access control is, in general, not possible.

Actually it is a bit more complicated. It must be the case that the “labels” must embeddable in a lattice. Defining this precisely is more complex than I want to get into now.