Information Flow

Source: Chapter 16 of *Computer Security: Art and Science* by Matt Bishop.

Information flow policies define how information may move throughout a system.

They typically are designed to protect the confidentiality or integrity of data:

- Confidentiality policies aim to prevent information from flowing to unauthorized users.

- Integrity policies aim to prevent untrustworthy data from flowing to trusted processes.

Access controls cannot fully constrain the flow of information in a system.

Both compile-time and runtime mechanisms can support the checking of information flows.

Entropy-Based Analysis

Intuitively, information flows from an object *x* to an object *y* if a sequence of commands *c* causes the information initially in *x* to affect that in *y*.

Entropy is an information-theoretic measure of the amount of uncertainty in a random variable.

Formally, the *entropy* of a random variable *X* with values in the set { $x_1, x_2, \ldots, x_n$ } is

$$H(X) = -\sum_{i=1}^{n} \Pr(X = x_i) \lg \Pr(X = x_i)$$

where lg *x* is the base-2 logarithm of *x*.

Example: If *M* is a random variable representing a message with *n* equally likely values, then

$$H(M) = -\sum_{i=1}^{n} \frac{1}{n} \lg \frac{1}{n} = -\lg n^{-1} = \lg n$$

That is, the entropy of *M* is the number of bits needed to represent it.

The *conditional entropy of X given that Y = y_j* is

$$H(X|Y = y_j) = -\sum_{i=1}^{n} \Pr(X = x_i|Y = y_j) \lg \Pr(X = x_i|Y = y_j)$$

The *conditional entropy of X given Y* is

$$H(X|Y) = -\sum_{j=1}^{m} \Pr(Y = y_j) \left[\sum_{i=1}^{n} \Pr(X = x_i|Y = y_j) \lg \Pr(X = x_i|Y = y_j) \right]$$

Let *c* be a sequence of commands taking a system from state *s* to another state *t*. Let *x* and *y* be objects in the system.

We assume that *x* exists when the system is in state *s* and has the value x_s, and we assume that *y* exists in state *t* with value y_t.

If *y* exists in state *s*, it has value y_s.

Definition. The command sequence *c* causes a *flow of information* from *x* to *y* if $H(x_s \mid y_t) < H(x_s \mid y_s)$.

This states that information flows from *x* to *y* if the value of *y* after the commands allows one to deduce information about the value of *x* before the commands were run.

Example: With the statement

```
y = x;
```

we have $H(\mathbf{x}_s \mid \mathbf{y}_t) = 0$.

Example: Consider the statement

```
x = y + z;
```

Let **y** take any of the integer values from 0 to 7 with equal probability.

Let **z** take the value 1 with probability 0.5 and the values 2 and 3 with probability 0.25 each.

Let *s* be the state before this operation is executed, and let *t* be the state immediately after it is executed.

Then $H(\mathbf{y}_s) = H(\mathbf{y}_t) = 3$ and $H(\mathbf{z}_s) = H(\mathbf{z}_t) = 1.5$.

Once the value of $\mathbf{x}_t$ is known, $\mathbf{y}_s$ can assume at most three values, so $H(\mathbf{y}_s \mid \mathbf{x}_t) < \lg 3 = 1.58$.

Similar results hold for $H(\mathbf{z}_s \mid \mathbf{x}_t)$.

Example: Consider a program in which **x** and **y** are integers that may be either 0 or 1. The statement

```
if (x == 1) y = 0;
else y = 1;
```

does not explicitly assign the value of **x** to **y**.

Assume that **x** is equally likely to be 0 or 1. Then $H(\mathbf{x}_s) = 1$.

But $H(\mathbf{x}_s \mid \mathbf{y}_t) = 0$, because if **y** is 0 then **x** is 1 and vice versa.

Hence, $H(\mathbf{x}_s \mid \mathbf{y}_t) = 0 < H(\mathbf{x}_s \mid \mathbf{y}_s) = H(\mathbf{x}_s) = 1$.

Thus, information flows from **x** to **y**.

Definition. An *implicit flow of information* occurs when information flows from x to y without an explicit assignment of the form $y = f(x)$, where $f(x)$ is an expression involving the variable x.

Note that an implicit information flow involves conditional execution of code.

Information Flow Models and Mechanisms

An *information flow policy* is a security policy that describes the *paths* along which information is authorized to flow.

An information flow model associates a *label*, representing a *security class*, with information and with entities containing that information.

Let x be a program variable. Then $\underline{x}$ denotes the security class of x.

The notation $\underline{x} \leq \underline{y}$ means that information is *permitted to flow* from class $\underline{x}$ to class $\underline{y}$.

Information Flow Policies

Denning identified two requirements for information flow policies:

- Information should be able to flow freely among members of a single security class (*reflexivity*).

- If members of class $\underline{y}$ can read information from class $\underline{x}$ they can save the information in variables of class $\underline{y}$. Then if members of class $\underline{z}$ can read information from $\underline{y}$, they can effectively read information from $\underline{x}$ (*transitivity*).

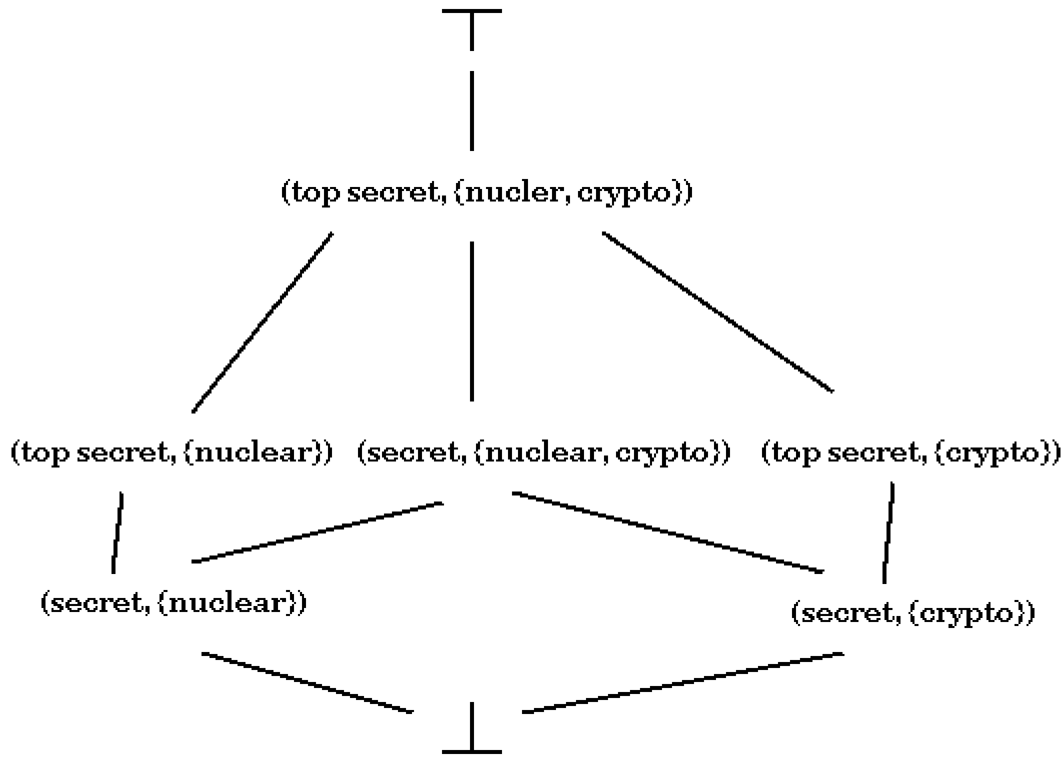
The *Bell and La Padula* model exhibits both of these characteristics.

The set of categories of this model form a *lattice* under the operation $\subseteq$:

- $\subseteq$ is *reflexive*, *antisymmetric*, and *transitive*.

- For every pair of categories C and D, $C \cap D$ is a *greatest lower bound* (GLB) and $C \cup D$ is a *least upper bound* (LUB).

Example:



⊤
(top secret, {nucler, crypto})
(top secret, {nuclear})
(secret, {nuclear, crypto})
(top secret, {crypto})
(secret, {nuclear})
(secret, {crypto})
⊥

However, in some cases, transitivity is *undesirable*.

Example: Suppose that Betty is a confidante of Anne, and that Cathy is a confidante of Betty.

If Anne tells Betty that she is having an affair with Cathy's husband, she does not want this information to flow to Cathy.

Formally, an *information flow policy* is a triple $I = (SC_I, \leq_I, join_I)$, where SC_I is a set of security classes, $\leq_I$ is an ordering relation on SC_I, and $join_I$ combines two elements of SC_I.

Example: For the Bell and La Padula model, SC_I is the set of security compartments, $\leq_I$ is the relation *dom*, and $join_I$ is set union.

Confinement Flow Model

Foley presented a model of confinement flow.

Assume that an object can change security classes.

Definition: The *confinement flow model* is a 4-tuple $(I, O, confine, \to)$ in which:

- $I = (SC_I, \leq_I, join_I)$ is a lattice-based information flow policy.
- O is a set of entities.
- $\to : O \times O$ is a relation with $a \to b$ if and only if information can flow from a to b.
- For each $a \in O$, $confine(a)$ is a pair $(\underline{a}_L, \underline{a}_U) \in SC_I \times SC_I$, with $\underline{a}_L \leq_I \underline{a}_U$, and the interpretation that for $a \in O$, if $\underline{x} \leq_I \underline{a}_U$, information can flow from x to a, and if $\underline{a}_L \leq_I \underline{x}$, information can flow from a to x.

This means that $\underline{a}_L$ is the *lowest classification* of information allowed to *flow out of a*, and $\underline{a}_U$ is the *highest classification* of information allowed to *flow into a*.

We have

$$(\forall a, b \in O)[a \rightarrow b \Rightarrow \underline{a}_L \leq_I \underline{b}_U]$$

Example: Let a, b, $c \in O$. Define

confine(a) = [CONFIDENTIAL, CONFIDENTIAL]

confine(b) = [SECRET, SECRET]

confine(c) = [TOPSECRET, TOPSECRET]

The possible information flows are $a \rightarrow b$, $a \rightarrow c$, $b \rightarrow a$, $b \rightarrow c$, $c \rightarrow a$, and $c \rightarrow b$.

The only flows that are secure according to the confinement model are $a \rightarrow b$, $a \rightarrow c$, and $b \rightarrow c$.

Thus, transitivity holds.

Now consider x, y, and z. These variables can assume values of different classifications:

$confine(x)$ = [CONFIDENTIAL, CONFIDENTIAL]

$confine(y)$ = [SECRET, SECRET]

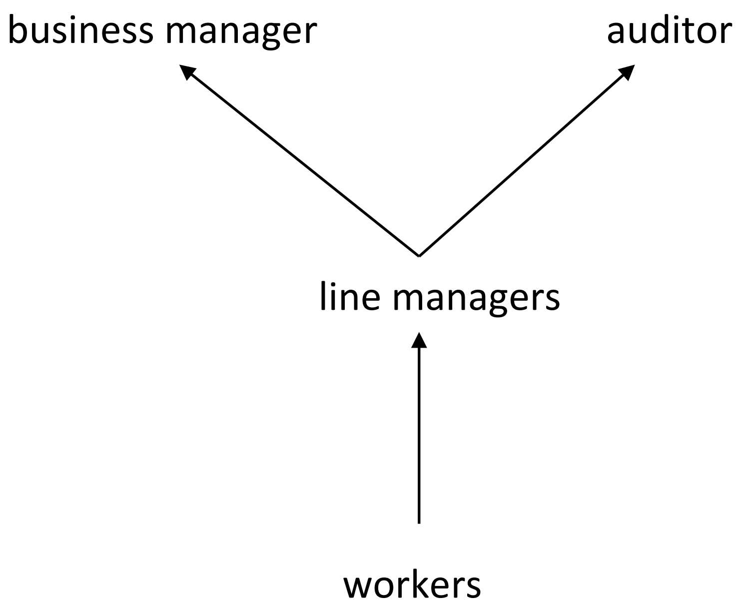
$confine(z)$ = [CONFIDENTIAL, TOPSECRET]

The only secure flows allowed are $x \rightarrow y$, $x \rightarrow z$, $y \rightarrow z$, and $z \rightarrow x$.

Information cannot legally flow from y to x, so transitivity fails.

Transitive Non-Lattice Information Flow Policies

Example: Consider the information flow policy described by the following diagram:



Workers report to line managers.

Line managers report to two superiors: a business manager and an auditor.

Since the latter do not report to an overall superior, the information flow relations don't form a lattice.

The company described above forms a "quasi-ordered set".

Definition. A *quasi-orderered set* $Q = (S_Q, \leq_Q)$ is a set S_Q and a relation $\leq_Q$ defined on S_Q such that the relation is both reflexive and transitive.

It is possible to define a lattice that includes a quasi-ordered set.

For all $x \in S_Q$, let $f(x) = \{ y \mid y \in S_Q \text{ and } y \leq_Q x \}$.

Define the set $S_{QP} = \{ f(x) \mid x \in S_Q \}$ and the relation $\leq_{QP} = \{ (x, y) \mid x, y \in S_{QP} \text{ and } x \subseteq y \}$.

Then S_{QP} is a partially ordered set under $\leq_{QP}$.

f preserves ordering, so $x \leq_Q y$ if and only if $f(x) \leq_{QP} f(y)$.

To turn S_{QP} into a lattice, add the sets S_Q and $\varnothing$ to it.

Define the upper bound

$$ub(x, y) = \{ z \mid z \in S_{QP} \text{ and } x \subseteq z \text{ and } y \subseteq z \}.$$

Then define the least upper bound $lub(x, y) = \cap ub(x, y)$.

Define the lower bound $lb(x, y)$ and the greatest lower bound $glb(x, y)$ similarly.

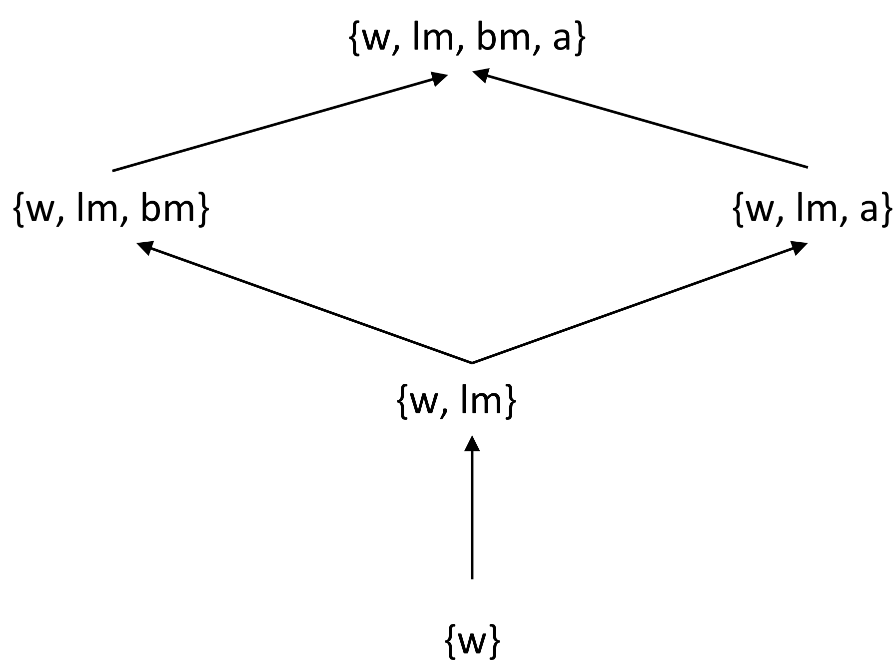
The structure $(S_{QP} \cup \{S_Q, \varnothing\}, \leq_{QP})$ is now a lattice.

Referring to the previous example:

- S_{QP} = { {w}, {w, lm}, {w, lm, bm}, {w, lm, a} }

- $\leq_{QP}$ is just the subset relation restricted to elements of S_{QP}.

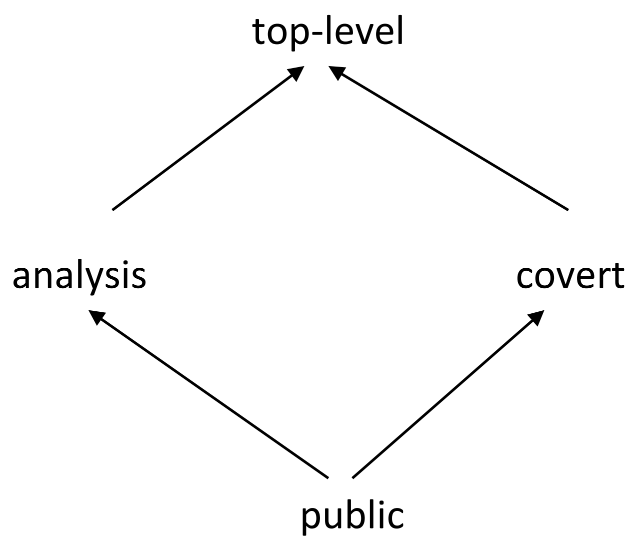
Lattice:



Non-transitive Information Flow Policies

Foley defined a procedure for building lattices from non-transitive systems.

Example: Consider a government agency whose information flow policy is illustrated by this diagram:



Suppose the agency has three types of entities: public relations officers (PRO), analysts (A), and spymasters (S).

Their accesses are confined to certain types of data as follows:

$confine$(PRO) = [public, analysis]

$confine$(A) = [analysis, top-level]

$confine$(S) = [covert, top-level]

According to the confinement flow model, PRO $\leq$ A, A $\leq$ PRO, PRO $\leq$ S, A $\leq$ S, and S $\leq$ A.

However, covert data cannot flow to PRO; S $\leq$ A and A $\leq$ PRO do *not* imply that S $\leq$ PRO.

Hence, the system is not transitive.

Definition. Let $R = (SC_R, \leq_R, join_R)$ represent a reflexive information flow policy. A *dual mapping* $(l_R(x), h_R(x))$ maps R to an ordered set $P = (S_P, \leq_P)$:

- $l_R(x) : SC_R \rightarrow S_P$ with $l_R(x) = \{ x \}$
- $h_R(x) : SC_R \rightarrow S_P$ with $h_R(x) = \{ y \mid y \in S_P \text{ and } y \leq_P x \}$

The relation $\leq_P$ indicates "subset" and the elements in S_P are subsets of SC_R.

The dual mapping is called *order preserving* if and only if

$$(\forall a, b \in SC_R)[a \leq_R b \Leftrightarrow l_R(a) \leq_P h_R(b)]$$

P is a partially ordered set, which can be transformed into a lattice by the method described previously.

Hence, without loss of generality, we assume it is a lattice.

Theorem. A dual mapping from a reflexive information flow policy R to an ordered set P is order preserving.

Proof. See Bishop.

Let *confine*(x) = $[\underline{x}_L, \underline{x}_U]$, and consider class $\underline{y}$.

Information can flow from x to an element of $\underline{y}$ if an only if $\underline{x}_L \leq_R \underline{y}$, or $l_R(\underline{x}_L) \subseteq h_R(\underline{y})$.

Information can flow from an element of $\underline{y}$ to x if and only if $\underline{y} \leq_R \underline{x}_U$, or $l_R(\underline{y}) \subseteq h_R(\underline{x}_U)$.

Example: Consider again the government agency and policy described earlier.

Call this policy R. We have the following relationships among the security classes:

public $\leq_R$ public	
public $\leq_R$ analysis	analysis $\leq_R$ analysis
public $\leq_R$ covert	covert $\leq_R$ covert
public $\leq_R$ top-level	covert $\leq_R$ top-level
analysis $\leq_R$ top-level	top-level $\leq_R$ top-level

The dual mapping elements l_R and h_R are:

l_R(public) = {public}	h_R(public) = {public}
l_R(analysis) = {analysis}	h_R(analysis) = {public, analysis}
l_R(covert) = {covert}	h_R(covert) = {public, covert}
l_R(top-level) = {top-level}	h_R(top-level) = {public, analysis, covert, top-level}

Let *p*, *a*, and *s* be entities of the types PRO, A, and S, respectively.

In terms of *P*, they are confined as follows:

confine(*p*) = [{public}, {public, analysis}]

confine(*a*) = [{analysis}, {public, analysis, covert, top-level}]

confine() = [{covert}, {public, analysis, covert, top-level}]

Thus,

$p \rightarrow a$ because {public} $\subseteq$ {public, analysis, covert, top-level}

$a \rightarrow p$ because {analysis} $\subseteq$ {public, analysis}

$p \rightarrow s$ because {public} $\subseteq$ {public, analysis, covert, top-level}

$a \rightarrow s$ because {analysis} $\subseteq$ {public, analysis, covert, top-level}

$s \rightarrow a$ because {covert} $\subseteq$ {public, analysis, covert, top-level}

However, because {covert} $\not\subset$ {public, analysis}, information cannot flow from *s* to *p*, reflecting the lack of transitivity of the system.

Compiler-Based Mechanisms

Compiler-based mechanisms check that potential information flows throughout a program are authorized.

Such mechanisms are *conservative*:

- They may indicate that illegal information flows are possible when in fact they are not.

- They will not fail to indicate illegal information flows (of the types they address) that are possible.

Compiler-based mechanisms can be used to *certify* that a program does not violate an information flow policy.

Example: Consider the conditional statement

```
if (x == 1) y = a;
else y = b;
```

Information flows from **x** and **a** to **y** and from **b** to **y**.

If the policy states that $\underline{\mathbf{a}} \le \underline{\mathbf{y}}$, $\underline{\mathbf{b}} \le \underline{\mathbf{y}}$, and $\underline{\mathbf{x}} \le \underline{\mathbf{y}}$, then the information flow can be certified as secure.

However, if $\underline{\mathbf{a}} \le \underline{\mathbf{y}}$ only when some other variable $z = 1$, a compiler-based mechanism generally cannot certify the statement.

Declarations

The security classes of variables must be declared, for the benefit of the checking mechanism.

Each variable might be assigned exactly one security class.

We will permit a *set of classes* to be declared, e.g.,

> *x*: *integer* **class** { *A*, *B* }

This indicates that *x* is an integer variable and that data from security classes *A* and *B* may flow into *x*.

The classes are assigned *statically*.

In terms of a lattice, this means that $lub\{ A, B \} \leq \underline{x}$.

Two distinguished classes, *Low* and *High*, represent the greatest lower bound and least upper bound, respectively, of the lattice.

All constants are of class *Low*.

Assignment Statements

An assignment statement has the form

$$y = f(x_1, \ldots, x_n)$$

where y and $x_1, \ldots, x_n$ are variables and f is some function of those variables.

Information potentially flows from each of the x_i's to y.

Hence, the requirement for the information flow to be secure is:

$$lub\{ \underline{x}_1, \ldots, \underline{x}_n \} \leq \underline{y}$$

Example: Consider the statement

```
x = y + z;
```

For this statement to be secure we must have $lub\{\underline{\mathbf{y}}, \underline{\mathbf{z}}\} \leq \underline{\mathbf{x}}$.

Compound Statements

A compound statement has the form

$$
\begin{array}{l}
\{ \\
\quad S_1; \\
\quad \ldots \\
\quad S_n; \\
\}
\end{array}
$$

where each of the S_i's is a statement.

If the information flow in each of the statements is secure, the information flow in the compound statement is secure.

Example: Consider the statements

```
{
    x = y + z;
    a = b * c – x;
}
```

The requirements for secure information flow are $lub\{\underline{\mathbf{y}}, \underline{\mathbf{z}}\} \le \underline{\mathbf{x}}$ and $lub\{\underline{\mathbf{b}}, \underline{\mathbf{c}}, \underline{\mathbf{x}}\} \le \underline{\mathbf{a}}$.

Conditional Statements

A conditional statement has the form

if $f(x_1, \ldots, x_n)$
    S_1;
else
    S_2;

where $x_1, \ldots, x_n$ are variables and where f is some boolean function.

The requirements for the information flow to be secure are that S_1 and S_2 are secure and that

$$lub\{\underline{x}_1, \ldots, \underline{x}_n\} \leq glb\{\underline{y} \mid y \text{ is a target of an assignment in } S_1 \text{ or } S_2\}$$

Example: Consider the statements

```
if (x + y < z)
   a = b;
else
   d = b * c – x;
```

Then the requirements for the information flow to be secure are that:

- $\underline{\mathbf{b}} \leq \underline{\mathbf{a}}$
- $lub\{\underline{\mathbf{b}}, \underline{\mathbf{c}}, \underline{\mathbf{x}}\} \leq \underline{\mathbf{d}}$
- $lub\{\underline{\mathbf{x}}, \underline{\mathbf{y}}, \underline{\mathbf{z}}\} \leq glb\{\underline{\mathbf{a}}, \underline{\mathbf{d}}\}$

Iterative Statements

Consider a loop statement of the form

while $f(x_1, \ldots, x_n)$ S;

The requirements for the information flow to be secure are

- The loop terminates.
- S is secure.
- $lub\{\underline{x}_1, \ldots, \underline{x}_n\} \leq glb\{\underline{y} \mid y$ is the target of an assignment in $S\}$

Example: Consider the loop

```
while (i < n) {
   a[i] = b[i];
   i = i + 1;
}
```

This loop terminates.

The loop body is secure if $\underline{i} \leq \underline{a[i]}$, $\underline{b[i]} \leq \underline{a[i]}$, and $\underline{i} \leq \underline{i}$, that is, if $lub\{\underline{i}, \underline{b[i]}\} \leq \underline{a[i]}$.

The flows from the loop condition into the loop body are secure if $lub\{\underline{i}, \underline{n}\} \leq glb\{\underline{a[i]}, \underline{i}\}$.