Incremental Relabeling

- A value may be assigned to a variable only if the relabeling that occurs at this point is a restriction, a relabeling in which the set of readers permitted by the new label is in all contexts a subset of the readers permitted by the original.
- For example, if the two labels differ only in a single policy, and the readers of that policy in the label of the variable are a subset of the readers of that policy in the label of the value.
  - Eg, a relabeling from \( \{ \text{bob} : \text{amy, carl} \} \) to \( \{ \text{bob} : \text{amy} \} \) is a restriction because the set of readers allowed by bob becomes smaller.
- If a relabeling is a restriction, it is considered to be safe.

- Remove a reader. It is safe to remove a reader from some policy in the label, as just described.
- Add a policy. It is safe to add a new policy to a label; since all previous policies are still enforced, the label cannot become less restrictive.
- Add a reader. It is safe to add a reader \( \text{r'} \) to a policy if the policy already allows a reader \( \text{r} \) that \( \text{r'} \) acts for. This change is safe because if \( \text{r'} \) acts for \( \text{r} \), it is already effectively considered to be a reader: \( \text{r'} \) has all of the privileges of \( \text{r} \) anyway.
- Replace an owner. It is safe to replace a policy owner \( \text{o} \) with some principal \( \text{o'} \) that acts for \( \text{o} \). This change is safe because the new policy allows only processes that act for \( \text{o'} \) to weaken it through declassification, while the original policy also allows processes with the weaker authority of \( \text{o} \) to declassify it.

- Clearly, a change to a label in which a reader \( \text{r} \) is removed cannot make the changed policy any less restrictive, and therefore this change is safe.
  - Note that the removal of a reader does not necessarily make the policy more restrictive if there is another reader in the policy for which \( \text{r} \) acts.
Examples of policy relationships

• 1. \{amy : bob, carl\} ⊆ \{amy : carl\}
• 2. \{amy : bob\} ⊆ \{amy : \}
• 3. \{amy : manager\} ⊆ \{amy : carl\}
• 4. \{manager : bob\} ⊆ \{carl : bob\}
• 5. \{amy : carl\} ≰ \{amy : bob\}
• 6. \{amy : carl\} ≰ \{bob : carl\}
• 7. \{amy : manager\} ≰ \{amy : bob\}
• 8. \{manager : bob\} ≰ \{bob : bob\}
Complete Relabeling Rule

- \( o(I) \) denotes the owner of a policy \( I \)
- \( r(I) \) denotes the set of readers of \( I \)
- principal \( p_1 \) acts for a principal \( p_2 \) in the principal hierarchy in which the relabeling is being checked, we will write \( p_1 \succeq p_2 \). Given a policy \( I \), we can define a function \( R \) that yields the set of principals implicitly allowed as readers by that policy:

\[
R(I) = \{ p \mid \exists p' \in r(I) \ p \succeq p' \}
\]

This function can be used to define when one label is at most as restrictive as another (\( L_1 \sqsubseteq L_2 \)) and when one policy is at most as restrictive as (that is, covers) another (\( I \sqsubseteq J \)):

**Definition of the complete relabeling rule (\( \sqsubseteq \))**

\[
L_1 \sqsubseteq L_2 \equiv \forall I \in L_1 \ \exists J \in L_2 \ I \sqsubseteq J
\]

\[
I \sqsubseteq J \equiv o(J) \succeq o(I) \land R(J) \subseteq R(I)
\]

\[
\equiv o(J) \succeq o(I) \land \forall p' \in r(J) \ \exists p \in r(I) \ p' \succeq p
\]
Join Rule for labels

• Least restrictive set of policies that enforces all the policies in $L_1$ and $L_2$ is simply the union of the two set of policies.

• The least restrictive label is the least upper bound or join of $L_1$ and $L_2$, written as $L_1 \sqcup L_2$ ($\oplus$ also has been used to denote the join of two security classes)

Labels for Derived Values (Definition of $L_1 \sqcup L_2$)

$$L_1 \sqcup L_2 = L_1 \cup L_2$$
Join of labels

- \{amy : bob\} \sqcup \{amy : bob, carl\} is
- \{amy : bob; amy : bob, carl\},
\equiv \{amy : bob\},
- the second policy in the union is covered by the first, regardless of the principal hierarchy
Declassification

- Labels in this model contain information about the owners of labeled data, these owners can retain control over the dissemination of their data, and relax overly restrictive policies when appropriate. This second kind of relabeling is a selective form of declassification.
- Code running with the authority of a principal can declassify data by creating a copy in whose label a policy owned by that principal is relaxed. In the label of the copy, readers may be added to the reader set, or the policy may be removed entirely, which is effectively the same as adding all principals as readers in the policy.
- At any moment while executing a program, a process is authorized to act on behalf of some (possibly empty) set of principals, referred to as the authority of the process.
- A process may weaken or remove any policies owned by principals that are part of its authority. Therefore, the label L1 may be relabeled to L2 as long as L1 ⊑ L2 ⊔ LA, where LA is a label containing exactly the policies of the form {p :} for every principal p in the current authority.
Declassification

For all policies $J$ in $L_1$, there must be a policy $K$ in $L_2$ that is at least as restrictive.

- declassification rule has the intended effect as for policies $J$ in $L_1$ that are owned by a principal $p$ in the current authority, a more restrictive policy $K$ is found in $L_A$.
- For other policies $J$, the corresponding policy $K$ must be found in $L_2$, since the current authority does not have the power to weaken them.
- This shows that a label $L_1$ always may be declassified to a label that it could be relabeled to by restriction, because the relabeling condition $L_1 \sqsubseteq L_2 \sqcup L_A$ implies the declassification condition $L_1 \sqsubseteq L_2 \sqcup L_A$.

Relabeling by declassification

\[
L_A = \bigsqcup\{p : p \text{ in current authority}\}
\]

\[
L_1 \sqsubseteq L_2 \sqcup L_A
\]

\[
L_1 \text{ may be declassified to } L_2
\]
Channels

- Input and output channels allow data to enter and leave the domain in which the label rules are enforced.
- Channels are half-variables; like variables, they have an associated label and can be used as an information conduit. However, they provide only half the functionality that a variable provides: either input or output.
  - As with a variable, when a value is read from an input channel, the value acquires the label of the input channel.
  - Similarly, a value may be written to an output channel only if the label of the output channel is at least as restrictive as the label on the value; otherwise, an information leak is presumed to occur.
Authority

Info transmission

Policy Specification
• The authority to act as Preparer need not be possessed by the entire WebTax application, but only by the part that performs the final release of the tax form.

• By limiting this authority to a small portion of the application, the risk of accidental release of the database is reduced. Thus the WebTax application might have a small top-level routine that runs with the authority of Preparer, while the rest of its code runs with no authority.
Integrity Labels

DUAL

Endorsement
Decentralized declassification in PL (Jif)

• Jif can track information flow at the level of individual variables and perform most label checks, at compile time. It also has the luxury of relying on the underlying operating system for bootstrapping, storage, trusted input files, administration, etc.,
• Jif labels allow different principals to express their security concerns by specifying what other principals are allowed to read or write certain data.
• Language-based techniques largely avoid addressing many practical issues such as trust management, resource allocation, support for heterogeneous systems, and execution of arbitrary machine code.
Securing Distributed Systems with Information Control

• Build secure applications from mostly untrusted code by using information flow control to enforce data security

• Eg. Enforce data security policy when executing untrusted code with access to sensitive data;
  – an untrusted application may be able to read some sensitive data, but it should not be able to surreptitiously export this data from the system

  • Example: virus scanner: accesses all of user’s private data but should never reveal their contents to anyone else (Symantec 10.x Anti-virus wormhole (2006) placed millions at risk)
• Application is usually quite large
  • It uses third party libraries
  • It has access to entire user database

• Works properly if all the code is verified
  • Which is impossible
• Bugs in application can enable data stealing
  (PayMaxx app code exposed 100,000 users' SSNs)
How do we work in untrusted environments?

- Eliminating bug in all application is impossible
- Can user data be kept secure even if applications are malicious?

**YES:**
- Track flow of user's data through system
- Only send user's data to that user's browser
- No need to audit/understand application code

**HOW:** OS'es like Asbestos, HiStar, Flume

**Limitation:** works only on one machine
- Web applications need multiple machines for scale
Information Flow Categories

**Explicit Flow**

```c
function test (bool high)
    bool low;
    low = high;
```

**Implicit Flow**

```c
function test (bool high)
    bool low = 0;
    if high = 1
        low = 1;
```

**Covert Flow**

```c
function test (bool high)
    bool low = 0;
    while high = 0;
    low = 1;
```
Java + Information Flow (Jif) [8]

- **Written in Java and built using polyglot extensible Java compiler framework.**

- **Jif provides support for information flow control and access control imposed at compile-time or run-time.**

- **Jif compiler follows Decentralized Label Model and static labelling.**

- **Jif compiler checks the specified flow-rules among mutual distrusted agents and compile successfully if the program is secure.**
Explicit Flow

void test (bool {Alice:Alice} high)
{
    bool {} low;
    low = high; // Jifc will throw error as {Alice:Alice} as {}  
    high = low; // No issue
}

Explicit Flow

void test (bool {Alice:Alice} high)
{
    bool low;
    low = high; // No issue
}

Read as

\{ owner : Reader \}

Label is public

Equivalent label inferred by Jifc

If no label is mentioned a program is always flow safe.
void test (int {Alice:Alice} x) {
    int {Bob:Bob} y = 1;
    int z = 0;

    if ( x == 0 )
        z = 1;

    if ( z == 1 )
        y = 0;
}

Copy 1 program from Denning’s book [10]

**Following static labelling:**

- **Label of z is inferred in such a way that all the flows to and from z are secure.**
- **There is no way to label z so that information in x can flow to z or information in z can flow to y.**
- **Hence Jifc throws error and the program is insecure.**
Information Flows

• Channels - mechanisms for signalling information

• Explicit Flows:
  – X:=Y – Y flows to X

• Covert Channels - primary purpose is not information transfer

• Implicit Flow:
  h:= h mod 2;
  l:=0;
  if h=1 then l:=1
  else skip

• Does not leak the exact value of i to l, but it does leak some information about the value of h to l

• Someone observing l could tell whether h is negative or not.
Readers-Writers Labels

• Security requirements of practical applications are often stated / easily understood in terms of who can read / write information

• Observations:
  – information readable by $s_1$ and $s_2$, can-flow-to information readable only by $s_1$
  – information writable only by $s_1$, can-flow-to information writable by $s_1$ and $s_2$

• Readers and writers can be used as labels!!
RWFM Label Format

• (owner/authority, readers, writers)
  – First component is a single subject denoting
    • *owner* in case of an object label
    • *authority* in case of a subject label
  – Second component is a set of subjects denoting
    • permissible readers in case of an object label
    • subjects who can read all the objects that this subject can read in case of a subject label
  – Third component is a set of subjects denoting
    • permissible writers in case of an object label
    • subjects who can write all the objects that this subject can write in case of a subject label
Permissible Flows in RWFM

• Given any two RW classes \( RW_1 = (s_1, R_1, W_1) \) and \( RW_2 = (s_2, R_2, W_2) \), information is allowed to flow from \( RW_1 \) to \( RW_2 \), denoted \( RW_1 \leq RW_2 \) only if \( R_1 \supseteq R_2 \) and \( W_1 \subseteq W_2 \). Formally

\[
\frac{R_1 \supseteq R_2 \quad W_1 \subseteq W_2}{(s_1, R_1, W_1) \leq (s_2, R_2, W_2)}
\]
Join and Meet of RW Classes

• Let RW\(_1\) = (s\(_1\), R\(_1\), W\(_1\)) and RW\(_2\) = (s\(_2\), R\(_2\), W\(_2\)), be any two RW classes. Their join (\(\oplus\)) and meet (\(\otimes\)) are defined as follows:

\[
(s_1,R_1,W_1) \oplus (s_2,R_2,W_2) = (s_3,R_1 \cap R_2,W_1 \cup W_2)
\]

\[
(s_1,R_1,W_1) \otimes (s_2,R_2,W_2) = (s_3,R_1 \cup R_2,W_1 \cap W_2)
\]
RW Classes form a Bounded Pre-Lattice

• **Prop**: The relation $\leq$ on RW classes is reflexive and transitive i.e., a **pre-order**

• **Theorem**: The set of all RW classes $\mathcal{SC}_{RW} = S \times 2^S \times 2^S$, together with the ordering $\leq$, join $\oplus$ and meet $\otimes$ form a **bounded pre-lattice**. For $s \in S$, $(s, S, \emptyset)$ denotes a minimum element and $(s, \emptyset, S)$ denotes a maximum element.
Readers-Writers Flow Model

• Above theorem establishes the soundness of RW classes w.r.t. Denning’s model i.e., suitability of RW classes for studying information flow properties in a system

• Readers-Writers Flow Model (RWFM) is defined as a six-tuple \((S,O,SC_{RW},\leq_{RW},\oplus_{RW},\otimes_{RW})\), where S is the set of subjects and O is the set of objects in an information system, and \(SC_{RW},\leq_{RW},\oplus_{RW},\otimes_{RW}\) are as defined previously
Notation

• Flow model together with a labelling function defines an access policy
• Labelling function $\lambda : S \cup O \rightarrow SC_{RW}$
• $A_\lambda (e)$, $R_\lambda (e)$ and $W_\lambda (e)$ denote the first, second and third components of $\lambda (e)$
• $\lambda$ is omitted when clear from the context
• For a subject $s$, $A(s)=s$
Access Rules in RWFM

• Given a RWFM and functions A, R and W describing a labelling,
  – A subject s is allowed to read an object o if
    • $A(s) \in R(o)$ and $R(o) \supseteq R(s)$ and $W(o) \subseteq W(s)$
  – A subject s is allowed to write an object o if
    • $A(s) \in W(o)$ and $R(s) \supseteq R(o)$ and $W(s) \subseteq W(o)$

DAC            MAC                             DAC + MAC
Completeness of RWFM w.r.t. Denning

• **Theorem:** Given a Denning’s flow model $DFM = (S,O,SC,\leq,\oplus)$ and a policy $\lambda : S \cup O \rightarrow SC$, there exists a labelling $\lambda_{RW} : S \cup O \rightarrow SC_{RW}$, in the RWFM that enforces the same policy i.e.,

1. $s$ is permitted to read $o$ by Denning’s policy if and only if it is permitted by RW-policy
2. $s$ is permitted to write $o$ by Denning’s policy if and only if it is permitted by RW-policy
Illustrative Examples

Denning’s Policy

Readers-Writers Policy
Illustrative Examples (contd)

Denning’s Policy

Readers-Writers Policy
State of an Information System

• State of an information system is defined as the set of subjects and objects in the system together with their labels. Initial state
  – Objects and their labels as required for application
  – Each subject s starts with label \((s, *, \phi)\)

• Whenever a subject tries to perform an operation on an object, it may lead to a state change and will have to be permitted only if deemed safe
  – Read
  – Write
  – Create
  – Downgrade
  – Relabel
State Transitions in RWFM

- Subject s with label \((s_1,R_1,W_1)\) requests read access to an object o with label \((s_2,R_2,W_2)\)
  - If \(s_1 \in R_2\) then
    - relabel s to \((s_1,R_1 \cap R_2,W_1 \cup W_2)\) and ALLOW access
  - Else
    - DENY access
  - **POSSIBLE** state change (label of s may change)
State Transitions in RWFM

- Subject $s$, with label $(s, R, W_1)$, requests write access to an object $o$ with label $(s_2, R_2, W_2)$.
  - If $s_1 \in W_2$ and $R_1 \supseteq R_2$ and $W_1 \subseteq W_2$ then
    - ALLOW access
  - Else
    - DENY access

- NO state change
State Transitions in RWFM

• Subject s with label \((s,R)\) requests creation of an object o
  
  – create an object o and label it \((s,R,W \cup \{s\})\)

• DEFINITE state change (a new object is added to the system)
State Transitions in RWFM

- Subject \( s \) with label \((s_1, R_1, W_1)\) requests an object \( o \) with label \((s_2, R_2, W_2)\) to be downgraded to label \((s_3, R_3, W_3)\).

  - If \( s_1 \in R_2 \) and \( s_1 = s_2 = s_3 \) and \( W_1 = W_2 = W_3 \) and \( R_1 = R_2 \) and \( R_3 \supseteq R_2 \) and \( R_3 - R_2 \subseteq W_2 \) then
    - ALLOW
  - Else
    - DENY

- POSSIBLE state change (label of \( o \) may change)
State Transitions in RWFM

- Subject $s$ with label $(s_1, R_1, W_1)$ requests an object $o$ with label $(s_2, R_2, W_2)$ to be relabelled with $(s_3, R_3, W_3)$

  - If $s_1 \in R_2$ and $s_1 = s_2 = s_3$ and $W_2 \subseteq W_1$ and $W_3 = W_1 \cup \{s\}$ and $R_2 \supseteq R_1 \supseteq R_3$ then
    - ALLOW
  - Else
    - DENY

- POSSIBLE state change (label of $o$ may change)
Downgrading (Declassifying)

• For practical applications, adding readers (downgrading) to the result of a computation is essential for use by relevant parties

• Downgrading rules
  – only the owner of information may downgrade it
  – if a single source is responsible for the information, then readers that can be added is unrestricted
  – if multiple sources influenced the information, then only those who influenced it may be added as readers
Reasoning about Information Flow between Objects in RWFM (1)

- **Theorem 1**: Information in object $o_1$ with label $(s_1, R_1, W_1)$ cannot flow to object $o_2$ with label $(s_2, R_2, W_2)$ if any of the following conditions hold:

1. $R_1 = \emptyset$
2. $W_2 = \emptyset$
3. $W_1 \not\subseteq W_2$
4. $R_2 \not\subseteq (R_1 \cup W_1 \cup W_2)$
Reasoning about Information Flow between Objects in RWFM (2)

• **Theorem 2**: If \( R_2 \subseteq R_1 \) and \( R_1 \cap W_2 \neq \emptyset \), and none of the conditions in Theorem 1 hold, only a subject in \( R_1 \) can make information to flow from \( o_1 \) to \( o_2 \).

• **Theorem 3**: If \( R_2 \subseteq (R_1 \cup W_1) \) and \( (R_1 \cup W_1) \cap W_2 \neq \emptyset \), and none of the conditions in Theorems 1 and 2 hold, information can flow from \( o_1 \) to \( o_2 \) only as a result of a collusion between a subject in \( R_1 \) with a subject in \( W_1 \).

( help us identify the only possible culprits in the case of an info. flow.)
Reasoning about Information Flow between Objects in RWFM (3)

• **Theorem 4**: If none of the conditions in Theorems 1, 2 and 3 hold, information can flow from \( o_1 \) to \( o_2 \) only as a result of a collusion between a subject in \( R_1 \) with all the subjects in \( R_2 \cap W_2 \).
Information Flow between entities in RWFM

- **Theorem:** Given a Denning’s flow model DFM = \( (S,O,SC,\leq,\oplus) \) with a policy \( \lambda : S \cup O \rightarrow SC \), and the corresponding policy in the RWFM (constructed in the completeness theorem), the following holds: “information can flow from entity \( e_1 \) to entity \( e_2 \) under Denning’s policy if and only if it can flow without downgrading in the RW-policy”, where entity is either a subject or an object in the system.
Informally

• While the completeness theorem proved that “immediate info flows” (flows resulting due to a single operation by subjects) in a Denning’s policy can be simulated by the corresponding RW policy, this theorem says that all info flows (in single or multiple steps between not only a subject and an object, but between any two entities) in a Denning’s policy can be simulated in the RW policy modulo downgrading.
Relations among subjects in RWFM

• **Prop:** Let DFM with $\lambda$ be a Denning’s policy, and let A, R and W denote the corresponding labelling in the RWFM (constructed in the completeness theorem). For any two subjects $s_1$ and $s_2$, the following holds:

1. $s_1 \in R(s_2)$ if and only if $R(s_2) \supseteq R(s_1)$
2. $s_1 \in W(s_2)$ if and only if $W(s_1) \subseteq W(s_2)$
Subject dominance relations in RWFM

• Subject $s_1$ "read dominates" $s_2$, $s_2 \leq_R s_1$, if $s_1 \in R(s_2)$
• Subject $s_1$ "write dominates" $s_2$, $s_2 \leq_W s_1$, if $s_1 \in W(s_2)$
• Subject $s_1$ "information dominates" $s_2$, $s_2 \leq_I s_1$, if $s_2 \leq_R s_1$ and $s_1 \leq_W s_2$
• Theorem: All the dominance relations on subjects are reflexive and transitive (pre-order)
Principal hierarchy vs subject dominance

• The standard notion of principal hierarchy can be captured as follows
  – Given subjects $s_1$ and $s_2$, we say that $s_1$ dominates $s_2$ in the principal hierarchy written $s_2 \leq s_1$, if $s_2 \preceq_R s_1$ and $s_2 \preceq_W s_1$

• Considering the fact that information flows in opposite directions in reading and writing, we recommend that in the context of IFC, information dominance provides a better notion of subject superiority than principal hierarchy
Example-1
WebTax

• Bob provides his tax-data to a professional tax preparer, who computes Bob’s final tax form using a private database of rules for minimizing the tax payable and returns the final form to Bob

• Security requirements
  1. Bob requires that his tax-data remains confidential
  2. Preparer requires that his private database remains confidential
Example-1
WebTax

TD (B, {B, P}, {B})

DB (P, {P}, {P})

IR (P, {P}, {B, P})

FF (P, {B, P}, {B, P})

<table>
<thead>
<tr>
<th>TD</th>
<th>Tax-data</th>
<th>IR</th>
<th>Intermediate results</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>Database of tax optimization rules</td>
<td>FF</td>
<td>Final tax form</td>
</tr>
<tr>
<td></td>
<td>Flows-to</td>
<td></td>
<td>Downgraded-to</td>
</tr>
</tbody>
</table>
### Example-1

**WebTax**

<table>
<thead>
<tr>
<th></th>
<th>DLM</th>
<th>DC</th>
<th>RWFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD</td>
<td>{B: B}</td>
<td>(B, B)</td>
<td>(B, {B,P}, {B})</td>
</tr>
<tr>
<td>DB</td>
<td>{P: P}</td>
<td>(P, P)</td>
<td>(P, {P}, {P})</td>
</tr>
<tr>
<td>IR</td>
<td>{B: B; P: P}</td>
<td>(B∧P, B∨P)</td>
<td>(P, {P}, {B,P})</td>
</tr>
<tr>
<td>FF</td>
<td>{B: B}</td>
<td>(B, B∨P)</td>
<td>(P, {B,P}, {B,P})</td>
</tr>
</tbody>
</table>

- **DLM label format**: policies separated by ‘;’, where each policy is of the form ‘owner: readers’
- **DC label format**: ‘readers, writers’, where readers control confidentiality, writers control integrity
- **RWFM label format**: ‘owner, readers, writers’
# DLM, DC and RWFM Comparison

<table>
<thead>
<tr>
<th></th>
<th>DLM</th>
<th>DC</th>
<th>RWFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>only Readers</td>
<td>only Readers</td>
<td>Readers and Writers</td>
</tr>
<tr>
<td>Integrity</td>
<td>only Writers</td>
<td>only Writers</td>
<td>Readers and Writers</td>
</tr>
<tr>
<td>Downgrading (DAC)</td>
<td>Purely discretionary</td>
<td>Purely discretionary</td>
<td>Consistent with IFC (MAC)</td>
</tr>
<tr>
<td>Ownership</td>
<td>Explicit</td>
<td>Implicit</td>
<td>Explicit</td>
</tr>
<tr>
<td>Authority</td>
<td>Orthogonal to the label</td>
<td>Orthogonal to the label</td>
<td>Explicit in the label</td>
</tr>
</tbody>
</table>
# DLM, DC and RWFM Comparison

<table>
<thead>
<tr>
<th></th>
<th>DLM</th>
<th>DC</th>
<th>RWFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal hierarchy and Delegation</td>
<td>Orthogonal to the label</td>
<td>Orthogonal to the label</td>
<td>Embedded in the label</td>
</tr>
<tr>
<td>Bi-directional flow</td>
<td>Difficult</td>
<td>Difficult</td>
<td>Simple and Accurate</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Easy</td>
</tr>
<tr>
<td>Label size</td>
<td>Moderate to Large</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>No. of labels</td>
<td>Large</td>
<td>Large</td>
<td>Small (as required by the application)</td>
</tr>
</tbody>
</table>
Readers-Writers Label Model

Advantages

• Labels are intuitive / easy to understand
• Automatic extraction of labels from security requirements
• Efficient label manipulations
• Easy to verify / validate required security properties
Language-based Security with RWFM
Security Labels for Program Points

• Compute security labels (RWFM) of the intermediate program points for identifying information misuse
  – makes the connection between a program and its environment explicit by including the stakeholders and system resources and their access controls in the execution context
  – clean semantics of (implicit) downgrading using a novel program construct
Language Syntax

Int
Bool
Var
Glob
Prog

Principal to which a value is to be returned is explicitly mentioned – crucial for implicit downgrading

\[ e \]
\[ c ::= \text{skip} \mid x := e \mid c; c \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c \mid \text{return } x \text{ to } p \]

\[ P, V, G, p : \text{BEGIN } c \text{ END} \]
Defining Information Misuse

- (MISUSE) A program is said to MISUSE information, if there exists a command at a point ‘i’ in the program which when executed causes an information flow that is not a permissible flow at that program point, i.e., violates the underlying can-flow-to lattice $\lambda_i$

- (SAFE) A program is said to be SAFE if it does not misuse information at any point in the program
Notations and Assumptions

• ‘$S$’ denotes the set of all the principals
• A special variable called ‘$pc$’ (program counter) depicts the current stage of the computation
• Evaluation context is defined by tuple $[M, \lambda]$
  – $M$ denotes memory contents
  – $\lambda$ denotes the labelling
• Functions $A$, $R$ and $W$ return the first (admin), second (readers) and third (writers) components of a label respectively
Notations (1)

• Constants (integers, Booleans) are immutably labelled (−, S, {})

• Globals (files and other system resources) are mostly statically labelled (with the exception of possible downgrading for returning) – initial labels are provided by the underlying environment as per the application requirements

• All other variables are dynamically labelled
  – labels are automatically computed at every stage of the computation

Language-based security literature, in contrast, considers all the variables to be statically labelled with the burden of providing labels left to the programmer
Notations (2)

• Following functions are available, $F : c \rightarrow 2^{\text{Var}}$
  – $\text{VA}$ variables
  – $\text{NG}$ non-global variables
  – $\text{MV}$ modified variables
  – $\text{MG}$ modified globals
  – $\text{MNG}$ modified non-global variables
  – $\text{AG}$ accessed globals
Deriving a Labelled Program for Interpreting Information Misuse

• We define a semantic mapping for deriving a security labelled program for a given program

\[ \llbracket \cdot \rrbracket : P \rightarrow P^L \]
Non-globals including the pc are initialized to 0, and assigned a label indicating:

\[ c \text{ is evaluated further in the environment } (P,V,G,p) \text{ and execution context } [M',\lambda'] \]

\[
\forall x \in G \left[ M'(x) = M(x) \land \lambda'(x) = \lambda(x) \right] \quad \forall x \in (NG(c) \cup \{pc\}) \left[ \lambda'(x) = 0 \land \lambda'(x) = (p,S,\{p\}) \right]
\]

\[
\langle P,V,G,p: \text{BEGIN } c \text{ END, } M, \lambda \rangle \rightarrow \langle P,V,G,p \rangle \vdash \langle c, M', \lambda' \rangle
\]

If any of the conditions fail, the program is declared to MISUSE information

\[
\left( p \notin P \right) \lor \left( G \notin V \right) \lor \left( V \neq VA(c) \right) \lor \left( \exists x \in NG(c) \left[ p \notin R(\lambda(x)) \right] \right)
\]

\[
\langle P,V,G,p: \text{BEGIN } c \text{ END, } M, \lambda \rangle \rightarrow \langle \text{MISUSE, } M, \lambda \rangle
\]