A Security Model of Dynamic Labeling Providing a Tiered Approach to Verification

Simon Foley
Department of Computer Science
University College
Cork, Ireland

Li Gong & Xiaolei Qian
Computer Science Laboratory
SRI International
Menlo Park, California, USA

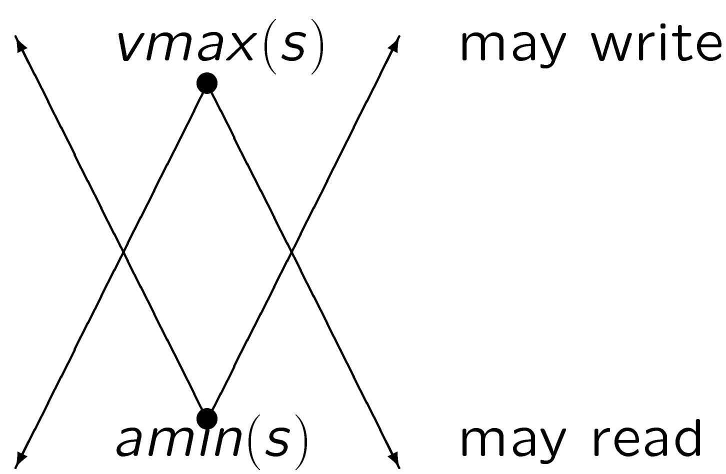
May 8, 1996

Introduction
Tiered Verification

Security Verification		*Approach*
	Applications & Operating System	Untrusted
Policy Axioms	Reconfigurable Security Requirements	Flow Policy & Relabel Rules
Model Axioms	Security Kernel	BLP-style MLS & Dynamic Relabeling

MLS Security Model
BLP with Partial Trust

- Flow Policy $(class, \leq)$
- Objects, Subjects
- Object Labels $lab(o)$
- Subject Labels
  $amin(s), vmax(s)$
- Model Access Axiom 1:



MLS Security Model
BLP & Relabeling Functions

Relabel function $f : (class \times class) \nrightarrow class$

$f(requester\_class, old\_class) = new\_class$

$f(r, a) = b$ means that when requested at class r,
information at class a may be relabeled as class b

Model Relabel Axioms 2&3:
Object and subject labels may change only according to the relabel functions.

Example

Flow Policy $(\mathit{class}, \leq)$

$$
\begin{tikzcd}
\texttt{topsecret} \\
\texttt{secret} \arrow[u] \\
\texttt{unclassified} \arrow[u]
\end{tikzcd}
$$

Relabel Functions
Upgrade & Downgrade

$$\texttt{up}, \texttt{down} : (\mathit{class} \times \mathit{class}) \pfun \mathit{class}$$

$$
\begin{aligned}
\texttt{up}(r, \texttt{unclassified}) &\mathrel{\widehat{=}} \texttt{secret} \\
\texttt{up}(r, \texttt{secret}) &\mathrel{\widehat{=}} \texttt{topsecret}
\end{aligned}
$$

$$
\begin{aligned}
\texttt{down}(r, \texttt{secret}) &\mathrel{\widehat{=}} \texttt{unclassified} \\
\texttt{down}(r, \texttt{topsecret}) &\mathrel{\widehat{=}} \texttt{secret}
\end{aligned}
$$

Relabel Macros

Relabel functions are defined in terms of classes (labels)

Relabel macros provide a system interface to relabel functions

```
/*
 * Upgrade object o
 */
up_object (subject s, object o){
  if pre(up(amin(s),lab(o)))
    relabelO(o,up);
}/*up_object*/
/*
 * Downgrade object o
 */
down_object (subject s, object o){
  if pre(down(amin(s),lab(o)))
    relabelO(o,down);
}/*down_object*/
```

Tiered Verification
System Interfaces

Security kernel provides security primitives

`relabel()`, `delete()`, etc.

Security Kernel Programming Interface

(SKPI)

Relabel macros provide applications and
operating system
with reconfigurable security primitives

`up_object`, `down_object`

Security Application Programming Interface

(SAPI)

Tiered Verification

Security Interface		*Security Verification*
(SAPI)	Applications & Operating System	Untrusted
`up-object`, `down-object` *(SKPI)*	Flow Policy, Subject/Object Labels, Relabel Functions & Relabel Macros	?
`relabel0`,...	Security Kernel	Model Axioms 1,2,3

Upgrade-from-Below Relabel Policies

Upgrade requested at, or below, class of label to be upgraded

$$
\begin{array}{ll}
\mathtt{up} : (\mathit{class} \times \mathit{class}) \nrightarrow \mathit{class} & \\
\mathtt{up}(\mathtt{unclassified}, \mathtt{unclassified}) & \widehat{=}\ \mathtt{secret} \\
\mathtt{up}(\mathtt{unclassified}, \mathtt{secret}) & \widehat{=}\ \mathtt{topsecret} \\
\mathtt{up}(\mathtt{secret}, \mathtt{secret}) & \widehat{=}\ \mathtt{topsecret}
\end{array}
$$

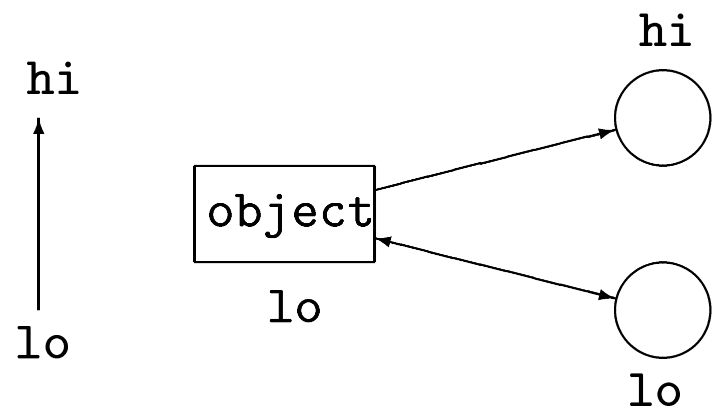
```
/*
 * Upgrade object o
 */
up_object (subject s, object o){
  if pre(up(amin(s),lab(o)))
    relabelO(o,up);
}/*up_object*/
```

Tiered Verification
Upgrade-from-Below Relabel Policies

Security Interface		*Security Verification*
(SAPI)	Applications & Operating System	
`up-object` *(SKPI)*	Reconfigurable Security Requirements	Relabel Functions are Upgrade-from-Below
`relabelO,...`	Security Kernel	Model Axioms 1,2,3

Can express many security requirements as upgrade-from-below policies

Example: Mark-for-Delete



Reference from **hi** to **lo** object.
Mark **lo** level object so that it is upgraded to **hi** when deleted by low level user

Approaches

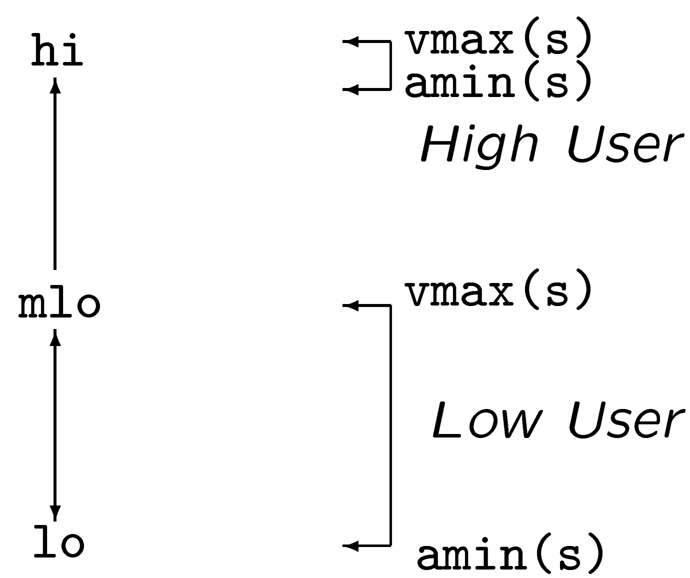
- Design & verify special kernel support
- Express requirement as a relabel policy

Example: Mark-for-Delete Relabel Policy: Requests made from Below

Relabel Functions | Flow Policy | Bindings

$$\mathtt{mark}(\mathtt{lo},\mathtt{lo}) \mathrel{\widehat{=}} \mathtt{mlo}$$

$$\mathtt{mdel}(\mathtt{lo},\mathtt{mlo}) \mathrel{\widehat{=}} \mathtt{hi}$$



Example: Mark-for-Delete Relabel Macros

```
/*
 * Mark low object o
 */
Mark (subject s, object o){
  if pre(mark(amin(s),lab(o)))
    relabelO(o,mark);
}/*Mark*/
/*
 * Delete object o
 */
MDelete (subject s, object o){
  if pre(mdel(amin(s),lab(o)))
    relabelO(o,mdel);
  else
    deleteO(o);
}/*MDelete*/
```

Tiered Verification
Mark-for-Delete (from Below)

Security Interface		*Security Verification*
(SAPI)	Applications & Operating System	
`Mark`, `MDelete` *(SKPI)*	Reconfigurable, Security Policy	Relabel Functions are Upgrade-from-Below
`relabel0`,`delete0`	Security Kernel	Model Axioms 1,2,3

Example: Mark-for-Delete
Mark Requested from Above

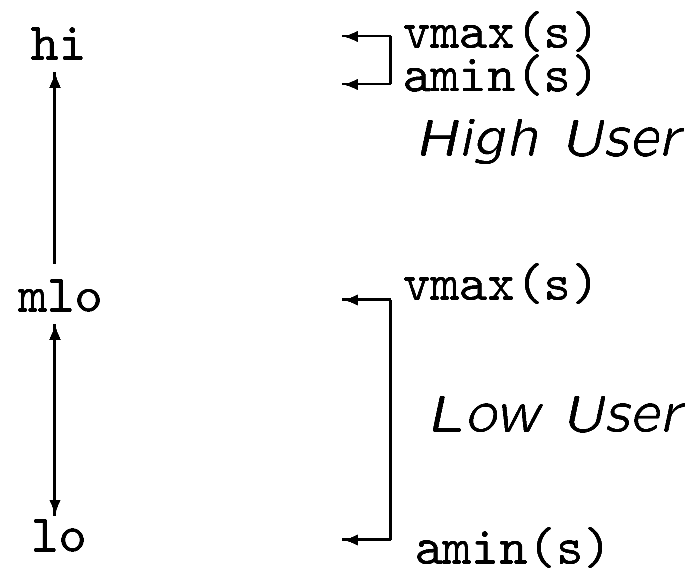
User at `hi` may wish to mark `lo` object

Relabel Functions | Flow Policy | Bindings

$$\mathtt{mark}(\mathtt{lo},\mathtt{lo}) \;\widehat{=}\; \mathtt{mlo}$$

$$\mathtt{mark}(\mathtt{hi},\mathtt{lo}) \;\widehat{=}\; \mathtt{mlo}$$

$$\mathtt{mdel}(\mathtt{lo},\mathtt{mlo}) \;\widehat{=}\; \mathtt{hi}$$



Covert Channel: user at `hi` either marks or not

Mark-for-Delete: Close Channel Secure Canonical Upgrade Policies (SCUP)

Security Kernel Requirement:
ensure that low user cannot distinguish between labels `lo` and `mlo`.

Subject Binding Requirement (Axiom 4,5):
relabeling an object from `lo` to `mlo` may not be detected by low user.
(low user must be bound to [`lo`,`mlo`])

Relabel Function Requirement (SCUP 1,2,3):
no sequence of high relabel requests can visibly interfere with a low label.
(`mark` may relabel only `lo` as `mlo`)

Tiered Verification
Secure Canonical Upgrade Policies

Security Interface		*Security Verification*
(SAPI)	Applications & Operating System	
`Mark`, `MDelete` *(SKPI)*	Reconfigurable, Security Policy	SCUP 1,2,3 Axioms 4,5
`relabelO`,`deleteO`	Security Kernel	Model Axioms 1,2,3

Conclusions

Two tier verification of security (kernel & policy verification)

Program/verify security primitives for specific applications (SAPI)

Based on familiar MAC-style security model

Supports a wide range of security policies