Labels and Event Processes in the Asbestos Operating System

Petros Efsthahopoulos  Maxwell Krohn
Steve VandeBogart    Cliff Frey
David Ziegler        Eddie Kohler
David Mazieres       Frans Kaashoek
Robert Morris

UCLA       MIT       Standford/NYU
Secure Information Flow

Confidentiality: secret data does not flow to unauthorized principals
  “data flow”

Bell–LaPadula model
  Security levels: top secret; secret; confidential; unclassified

Information flows up, but not down!
Secure Information Flow

Subjects and objects are assigned levels
A subject can read an object iff the subject’s level $\geq$ the object’s level: no “read up”
   Person with top-secret clearance can read secret files

A subject can write an object only iff the subject’s level $\leq$ the object’s level: no “write down”
   Person with secret clearance can write top-secret files
Integrity [Biba 1977]

Information flow can be used to enforce integrity as well as confidentiality
Integrity is the dual of confidentiality

Levels: low-integrity (tainted); high-integrity
Tainted data should not flow to places that demand high-integrity data
Who Assign Levels?

A centralized version
A central authority assigns levels to subjects and objects

Decentralized Label Model [Myers and Liskov 2000]
Idea: each subject can control its own data independently

E.g., subject A: assign his files the “secret” level; assign subject B “top secret”, subject C “confidential”

Each subject can also declassify its own data
Asbestos: an OS with Mandatory Control of Information Flow

Labels
- Decentralized label model
  Asbestos kernel tracks labels and performs runtime label checking

Event processes
- Lightweight processes with thread-like semantics
- Targets the web-server application
Asbestos Labels

label: categories -> levels
Categories are called “handles” in the paper

Each category identifies a category of data

levels = {* , 0, 1, 2, 3}

Lattice operations
L1 v L2 iff L1(c) · L2(c) for all c

(L1 t L2)(c) = max (L1(c), L2(c))
Process Labels

Each process has two labels: send and receive labels

Send label

- Models its current taint level

Receive label

- Models its maximum taint label (AKA clearance)

Idea: a process may get more tainted as it receives more data, but it should be no greater than the clearance

P may send data to Q if $PS \leq QR$

After Q receives data, it is more tainted:

- $QS = QS \oplus PS$
Comments on Process Labels

- Process-level tainting: coarse grained
- E.g., if the process receives top-secret data, then the whole process’s send label becomes “top secret”
- Data sending automatically changes the receiver’s send label
- This is a covert storage channel
- Fixed in HiStar: the receiver raises its own label before receives a message
Confidentiality Example

Two users: U and V
Security goal: U’s data can flow to the terminal, but not V’s
A category $u_T$ for U’s files and $v_T$ for V’s files
U’s files has label $\{u_T 3, 2\}$; V’s files has label $\{v_T 3, 2\}$

Labels above prevent the followings
V’s shell cannot send data to U’s terminal
User u cannot read v’s files
Some Variation

As a result, U can read V’s data. But after U read V’s data, it loses the ability to communicate with UT.

“export protection”
Special Privilege: *

Declassification
E.g., declassify “top secret” files to “unclassified” so that everyone can read it

User V may want to share some of his files with U

Decentralized declassification: anyone with ownership on some data should be able to declassify it
Processes with * on a category does not get tainted even after it receives tainted data
What happens if $FSs = \{uT0, vT0, 1\}$?
Contamination Label

File server must taint $U$ with taint $uT^3$, and taint $V$ with taint $vT^3$

Solution: each message comes with an optional contamination level

$FS_s = \{uT^*, vT^*, 1\}$
$FS_R = \{uT^3, vT^3, 2\}$
Other Bits

Decontamination
A process with ownership on a category can also decontaminate the send label of other processes

Communication ports
Each port has a receive label that restricts the messages that are delivered to that port
Application: OKWS web server

Goal: isolating different users’ data from one another so that a compromised user cannot access other users’ data
HiStar, Flume

HiStar: very similar to Asbestos
   Fixed a covert channel in Asbestos
   Fixed some label-formulation errors in Asbestos

Flume: user-level enforcement of labels
What’s good about these systems?

Demonstrate how DIFC works in an OS
Coarse-grained, process-level DIFC seems efficient
  • Byte-level information flow is expensive
Not Organized