Mandatory Access Control

COS 316: Principles of Computer System Design

Amit Levy & Ravi Netravali
Last Times - Discretionary Access Control

Discretionary Access Control - [Access] controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

- Trusted Computer System Evaluation Criteria, 1985 (the “Orange Book”)

- Most prevalent form of access control system
- Access Control Lists
  - Restrict access to objects based on the identity of subjects
  - Subjects can pass object contents after reading
- Capabilities
  - Restrict access to objects based on possession of a capability
  - Capabilities can be passed directly to other subjects
  - Subjects can pass object contents after reading
Who enforces policy under DAC?

Legend
- FBI
- SNCC

What might go wrong?
Who enforces policy under DAC?

Legend
- FBI
- SNCC

Which components enforce policy?

- Only repository collaborators can read code from private repositories.
- Only repository collaborators can comment on repositories.
Who enforces policy under DAC?

Legend
- Trusted Computing Base

App

WWW

Guard

Repo 1

Repo 2

Only repository collaborators can comment on repositories.

Only repository collaborators can read code from private repositories.

Read repo 1 code

Write comment to repo 2
Limitations of Discretionary Access Control

- Discretionary means *subjects* with access to *objects* can propagate information:
  - UNIX process reads `~/.ssh/ida_rsa` and writes output to public log
  - In UNIX, owners determine read/write/execute access for themselves, group, and “other”
  - Subject can pass capabilities to anyone
  - Can’t (trivially) revoke capabilities

- This is *one* reason it is sufficient to compromise a single high privilege application, not whole system, in order to extract private data
The non-interference property

Informally:

A program is non-interferent if it’s transformations of data in low security domains (*low*) are not influenced by data in higher security domains (*high*)
The non-interference property

\(M\), a memory state including low and high memory, \(M_H\) and \(M_L\), respectively

\(P: (M) \rightarrow M^*\), a non-interference program execution over a memory state resulting in a new memory state, if:

\[ \forall M_1, M_2 \text{ s.t. } M_{1_L} = M_{2_L} \]
\hfill \land \quad P(M_1) \rightarrow M_{1^*} \quad \land \quad P(M_2) \rightarrow M_{2^*} \quad \Rightarrow \quad M_{1^*_L} = M_{2^*_L} \]
Enforcing Non-Interference with DAC

Discretionary Access Control policies can enforce non-interference by completely partitioning the system.
Enforcing Non-Interference with DAC

Discretionary Access Control policies can enforce non-interference by completely partitioning the system, or with careful, static sharing.
Mandatory Access Control (MAC)

- Goal: data secrecy & integrity don’t rely on trusting applications at all
- All resource accesses governed by a global policy
- Subjects cannot change or circumvent global policy
- Typically policy articulated in terms of data sources and sinks
- E.g.
  - *label* data with its sensitivity
  - define permitted flows between labels
  - Permit operations as long as information flow rules are not violated
A simple security label lattice

Repo 1

Repo 2

Public

Flow not permitted!
Implementing MAC

There are very few MAC systems used in practice:

- SELinux - an extension to Linux originating from the NSA
  - Used in Android
- Mandatory Integrity Control - a Windows kernel subsystem limited to integrity
- TrustedBSD (in development)

But lots of research systems
Implementing MAC

One general approach:

● Assign a security label to object (file, network endpoint, console, etc)
● Assign a *floating* label to subjects (running processes)
   ○ “Floating” because it changes dynamically
● Whenever moving/copying data, check that source label can flow to sink label
● Allow subject to “raise” its floating label, but not to “lower” it
Permissible, because write *couldn’t* involve secret data
Permissible, because write could only involve data secret to Repo 2.
Prohibited, because write to Repo 1 could involve data secret to Repo 2
Prohibited, because write could involve data secret from **Repo 2** or **Repo 1**
Mandatory Access Control in Practice

- Dates back to at least 1983
  - Defined in the DoDs *Trusted Computer System Evaluation Criteria* (aka the Orange Book)

- Very powerful guarantee!
  - Security policies on data *do not* rely on application correctness

- Why is it not more prevalent?
Why isn’t MAC more prevalent?

- Complexity: implementing MAC can be hard to get right
- Performance: lattice checks can be slow
- Flexibility: by design, applications cannot get around security policy
- Simplicity: MAC is harder to administer