CIS 4930: Secure IoT

Lecture 8

Prof. Kaushal Kafle

Derived from slides by Adwait Nadkarni, William Enck, Micah Sherr and Patrick McDaniel

Principle of Least Privilege

A system should only provide those rights needed to perform the processes function and no more.

- Implication 1: you want to reduce the protection domain to the smallest possible set of objects
- Implication 2: you want to assign the minimal set of rights to each subject
- Caveat: of course, you need to provide enough rights and a large enough protection domain to get the job done.

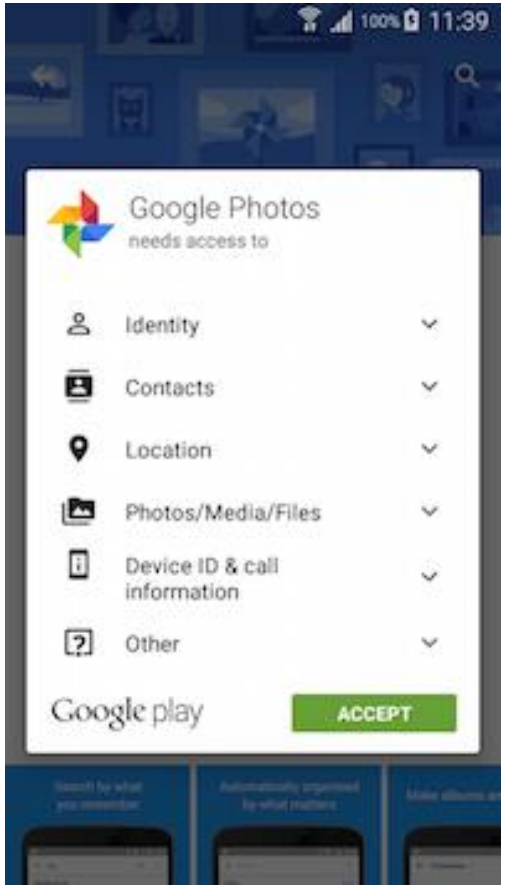
Least Privilege

- Limit permissions to those required and no more
  - Restrict privilege of the process of J to prevent leaks
    - Cannot R/W O3

	O_1	O_2	O_3
J	R	RW	-
S_2	-	R	-
S_3	-	R	RW

Least Privilege

- Pros:
  - Removes unnecessary permissions (avoid confused deputy?)
  - Ensures least permissions to carry out all functionalities.
- Cons:
  - Task execution can still conflict with security goals.
  - Guarantees secure policies?
  - No! Least privilege policies based on functions, not security.



Conflicting Goals

- Challenges of building a secure system
  - What are the *users*' goals?
  - What do *application developers* want?
  - What about the *data owners* (corporations/governments)?
  - What is the purpose of *system administrators*?
  - What about the requirements of *operating system designers*?
- Need a *satisfying* balance among these goals..

Access Control Administration

There are two central ways to specify a policy

- *Discretionary* - object "owners" define policy
  - Users have discretion over who has access to what objects and when (trusted users)
  - Canonical example: the UNIX filesystem
    - RWX assigned by file owners
- *Mandatory* - Environment enforces static policy
  - Access control policy defined by environment, user has no control over access control (untrusted users)
  - Canonical example: process labeling
    - System assigns labels for processes, objects, and a dominance calculus is used to evaluate rights

DAC vs. MAC

- **Discretionary Access Control**
  - User defines the access policy
  - Can pass rights onto other subjects (called *delegation*)
  - Their programs can pass their rights
    - Consider a Trojan horse (e.g., you get me to run your code in my system)
- **Mandatory Access Control**
  - System defines access policy
  - Subjects cannot pass rights
  - Subjects' programs cannot pass rights
    - Consider a Trojan horse here (e.g., you get me to run your code in my system)

DAC vs. MAC in Access Matrix

- Subjects:
  - DAC: users
  - MAC: labels
- Objects:
  - DAC: files, sockets, etc.
  - MAC: labels
- Operations:
  - Same
- Administration:
  - DAC: owner, copy flag, …
  - MAC: external, reboot
- MAC: largely static matrix;
- DAC: all can change

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

Safety Problem

- For a protection system
  - (ref mon, protection state, and administrative operations)
- Prove that any future state will not result in the leakage of an access right to an unauthorized user
  - Q: Why is this important?
- For most discretionary access control models,
  - Safety is *undecideable*
- Means that we need another way to prove safety
  - Restrict the model (no one uses)
  - Test incrementally (constraints)
- *How about MAC models?*

Sandboxing

- An execution environment for programs that contains a limited set of rights
  - A subset of your permissions (meet secrecy and integrity goals)
  - Cannot be changed by the running program (mandatory)

Case Study – Android UIDs

- Android is a *Linux-based system*
- Apps are security principles*, treated as users*
- Apps acquire *permissions* to access ...
- What separates apps from one another?
- What separates Apps from the kernel?
- What prevents apps from access to arbitrary storage?

Access Control Models

- What language should I use to express policy?
  - Access Control Model
- Oodles of these
  - Some specialize in secrecy
    - Bell-LaPadula
  - Some specialize in integrity
    - Clark-Wilson
  - Some focus on jobs
    - RBAC
  - Some specialize in least privilege
    - SELinux Type Enforcement

- Q: Why are there so many different models?

Information Flow Control

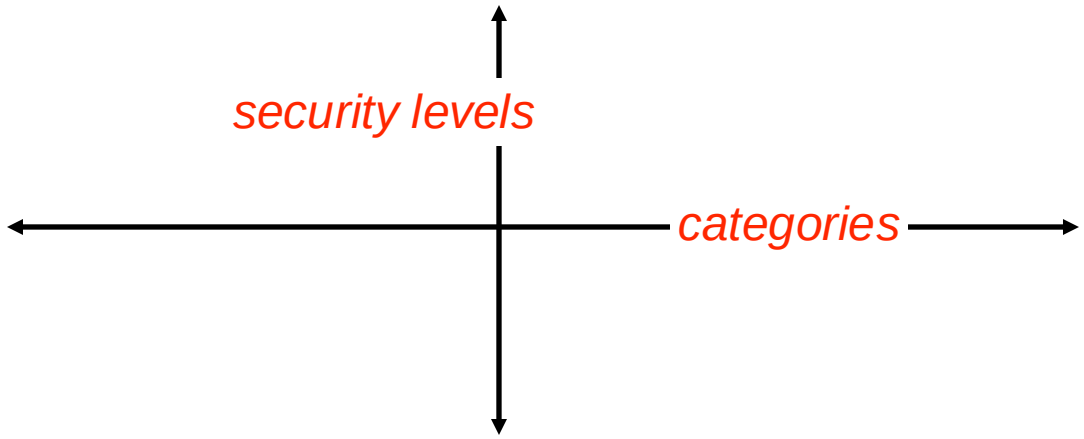
Information Flow Control

- Ensures *authorized* flow of *information/data* among system entities

Access Control Models	IFC models
• Regulates actions of subjects on objects • Concerned about *access* to certain resources within a system	• Regulates what info is being transferred between entities • Concerned about *data movement*

Multilevel Security

- A multi-level security system tags all object and subject with security tags classifying them in terms of sensitivity/access level.
  - We formulate policies based on these levels
  - We can also add other dimensions, called categories which horizontally partition the rights space (in a way similar to that as was done by roles)



US DoD Policy

- Used by the US military (and many others), the Lattice model uses MLS to define policy
- Levels:

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

- Categories (actually unbounded set)

NUC(lear), INTEL(igence), CRYPTO(graphy)

- Note that these levels are used for physical documents in the governments as well.

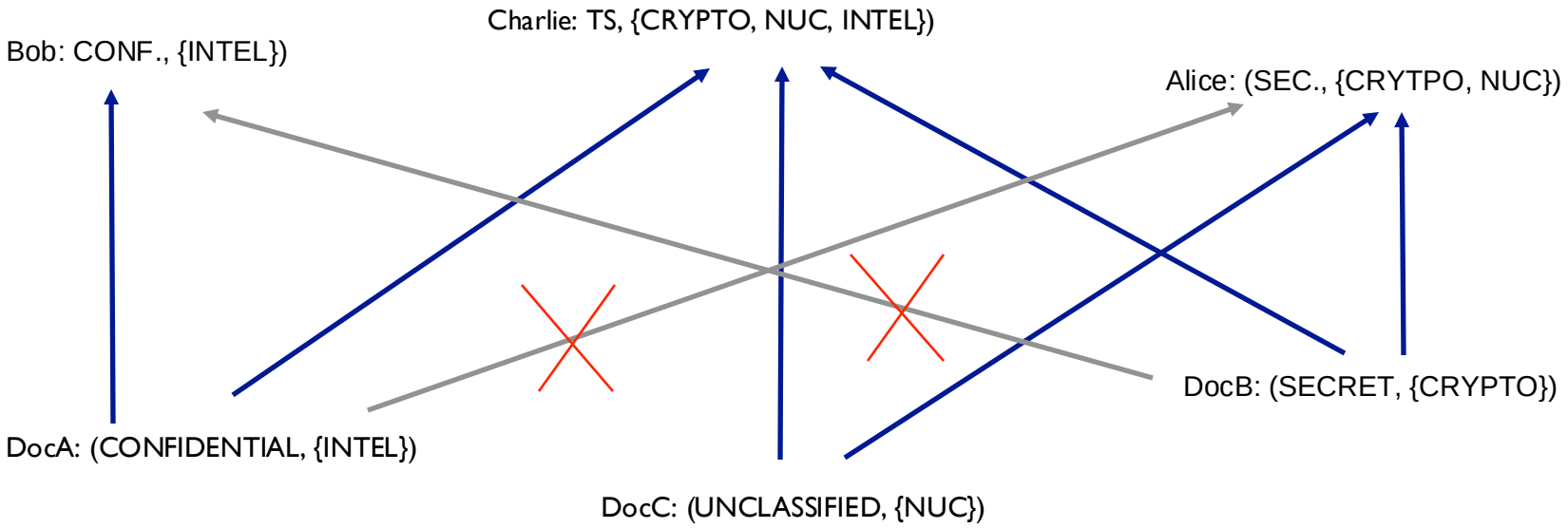
Assigning Security Levels

- All subjects are assigned clearance levels and compartments
  - Alice: (SECRET, {CRYTPO, NUC})
  - Bob: (CONFIDENTIAL, {INTEL})
  - Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL})

- All objects are assigned an access class
  - DocA: (CONFIDENTIAL, {INTEL})
  - DocB: (SECRET, {CRYPTO})
  - DocC: (UNCLASSIFIED, {NUC})

Evaluating Policy

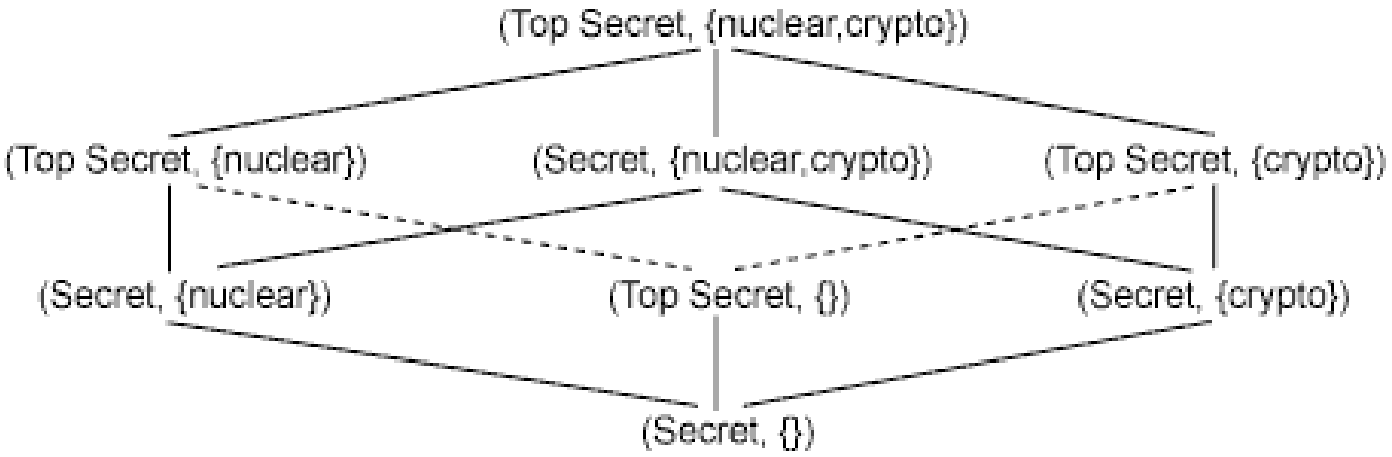
- Access is allowed if
- subject clearance level >= object sensitivity level *and* subject categories ⊇ object categories (*read down*)



Q: What would *write-up* be?

Bell-LaPadula (BLP) Model

- A Confidentiality MLS policy that enforces:
  - *Simple Security Policy*: a subject at specific classification level cannot read data with a higher classification level. This is short hand for "*no read up*".
  - ** (star) Property*: also known as the confinement property, states that subject at a specific classification cannot write data to a lower classification level. This is shorthand for "*no write down*".
- E.g., corporate hierarchies



How about integrity?

- Before: MLS considered who can "read" a document (confidentiality)
- Integrity considers who can "write" to a document
  - Thus, who can effect the integrity (content) of a document
  - Example: You may not care who can read DNS records, but you better care who writes to them!
- Biba defined a dual of secrecy for integrity
  - Goal: Do not depend on data from lower integrity principals
  - Flow permitted only from high to low integrity
  - User's integrity level must be above or equal to that of the file being modified.

Biba integrity

- Biba: User's integrity level must be above or equal to that of the file being modified.
  - Lattice policy with, "no read down, no write up"
    - Users can only *create* content at or *below* their own integrity level (a monk may write a prayer book that can be read by commoners, but not one to be read by a high priest).
    - Users can only *view* content at or *above* their own integrity level (a monk may read a book written by the high priest, but may not read a pamphlet written by a lowly commoner).

Biba (example)

- Which users can modify what documents?
  - Remember "*no read down, no write up*"

Charlie: (TS, {CRYPTO, NUC, INTEL})

Bob: (CONF., {INTEL})

Alice: (SEC., {CRYTPO, NUC})

?????

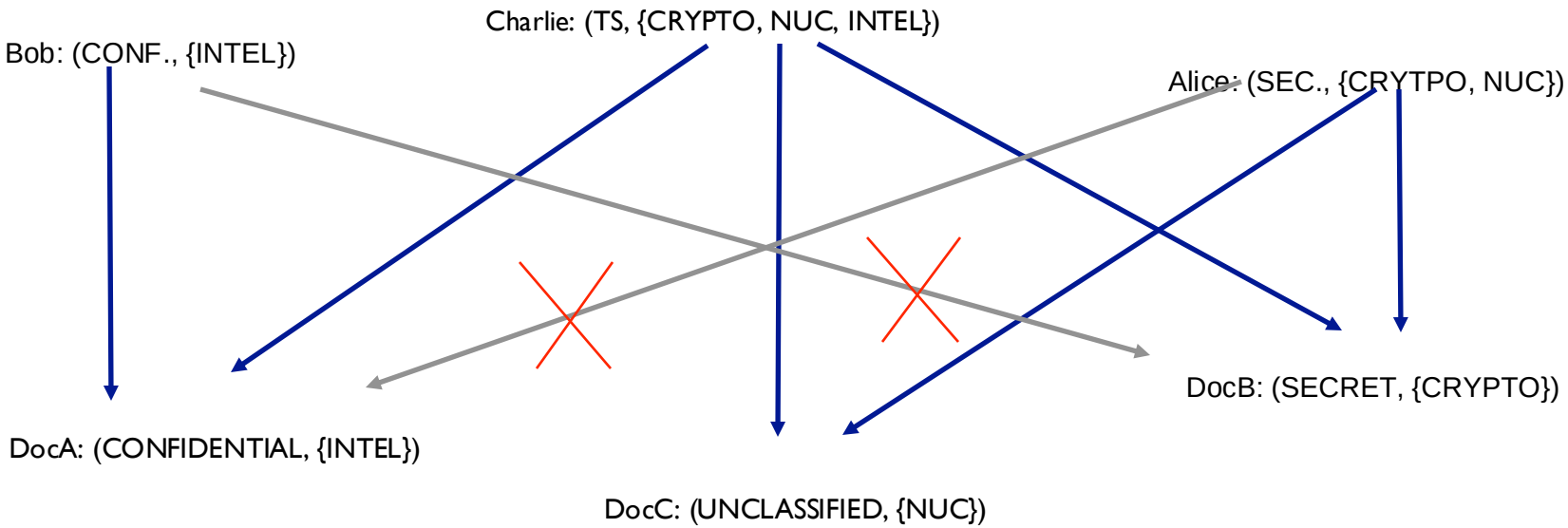
DocB: (SECRET, {CRYPTO})

DocA: (CONFIDENTIAL, {INTEL})

DocC: (UNCLASSIFIED, {NUC})

Biba (example)

- Which users can modify what documents?
  - Remember "*no read down, no write up*"



Biba - Guards

- What happens if the higher integrity user needs information from lower integrity file?
  - E.g., reading from network sockets
- Unauthorized under Biba!
  - Unless subject is *fully assured* to upgrade to high integrity or discard low integrity data
  - Done by '*guards*'

LOMAC

- Low-Water Mark integrity
  - Change integrity level based on actual dependencies
- Subject is initially at the highest integrity
  - But integrity level can change based on objects accessed
- Ultimately, subject has integrity of lowest object read
  - Example of "*self revocation*"

Integrity, Sewage, and Wine

- Mix a gallon of sewage and one drop of wine gives you?
- Mix a gallon of wine and one drop of sewage gives you?

Integrity is really a contaminant problem: you want to make sure your data is not contaminated with data of lower integrity.