CSE 543: Computer Security
Module: Mandatory Access Control

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
The Pennsylvania State University

Security Goals

- Secrecy
  - Don't allow reading by unauthorized subjects
  - Control where data can be written by authorized subjects
    - Why is this important?
- Integrity
  - Don't allow modification by unauthorized subjects
  - Don't allow dependence on lower integrity data/code
    - Why is this important?
  - What is "dependence"?
- Availability
  - The necessary function must run
  - Doesn't this conflict with above?

Trusted Processes

- Do you trust every process you run?

Trusted Processes

- Do you trust every process you run?
  - To not be malicious?

Trusted Processes

- Do you trust every process you run?
  - To not be malicious?
  - To not be compromised?

Secrecy

- Does the following protection state ensure the secrecy of J's private key in O_1 (i.e., S_2 and S_3 cannot read)?

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

Secrecy Threat

- Trojan Horse
  - *Some process of yours is going to give away your secret data*
    - Write your photos to the network

Integrity

- Does the following access matrix protect the integrity of J's public key file O_2?

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

Integrity Threat

- Untrusted Input
  - *Process reads untrusted input when expects input protected from adversaries*
    - Read a user-defined config file
    - Execute a log file
    - Admin executes untrusted programs

Access Control Administration

There are two central ways to manage a policy

1. 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
2. Mandatory - Environment defines policy
   - OS distributor and/or administrators define a system policy that cannot be modified by normal users (or their processes)
   - Typically, information flow policies are mandatory
     - More later…

Protection vs Security

- Protection
  - Secrecy and integrity met under *benign* processes
  - Protects against an error by a non-malicious entity
- Security
  - Secrecy and integrity met under *malicious* processes
  - Blocks against any malicious entity from performing unauthorized operations at all times
- Hence, For J:
  - Non-malicious processes shouldn't leak the private key by writing it to O_3
  - A malicious or compromised process may contain a Trojan horse that will write the private key to O_3

Is It Possible?

- For a protection system to enforce security?
  - No, and it was proven

Safety Problem (HRU 1976)

- For a protection system
  - (protection state and administrative operations)
- Prove that all future states 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 likes)
  - Test incrementally (constraints)
- Proven by Harrison, Ruzzo, and Ullman (CACM 1976)

Meaning

- We cannot design an access matrix policy for a UNIX protection system that we can prove will prevent an unauthorized access
  - Processes can modify the matrix
  - New files extend the matrix

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

Security Goals

- What security goals should access control policies describe?
  - Secrecy, Integrity, Availability
  - How are they balanced?
- Secrecy
  - Prevent leakage of X to unauthorized subjects
- Integrity
  - Prevent modification of Y by unauthorized subjects
- Availability (Functionality)
  - Enable required subjects to read X and write Y
- How do we balance such goals?

What Is Security?

- *In practice, security methods focus on security or functionality - but not both at the same time!*
- Security Is Foremost
  - Information Flow: No communication with untrusted
  - Advantage: Focus is security
  - Disadvantage: May prevent required functionality
- Restrict based on Functionality
  - Least Privilege: Only rights needed to execute
  - Advantage: Enables required functionality
  - Disadvantage: May not block all attack paths
- Let's look at the two common approaches
  - Least Privilege and Information Flow

Principle of Least Privilege

- Implication 1: you want to limit the process to the smallest possible set of objects
- Implication 2: you want to assign the minimal set of operations to each object

A system should only provide those privileges needed to perform the processes' functions and no more.

- Caveat: of course, you need to provide enough permissions to get the job done.

Least Privilege

- Limit permissions to those required and no more
  - Consider three processes for user J. J_1-J_3 must use the permissions below
    - What is the impact of the secrecy of O_1?
    - Restrict privilege of the process J1 to prevent leaks

	O_1	O_2	O_3
J_1	R	RW	-
J_2	-	R	-
J_3	-	R	RW

Least Privilege

- Can least privilege prevent attacks?
  - Trojan horse
  - Untrusted input

Least Privilege

- Can least privilege prevent attacks?
  - Trojan horse
  - Untrusted input
- Some. No guarantee such attacks are not possible

Verifying Least Privilege

- Most real-world access control policies have the goal of achieving (approximating) least privilege
- What does it mean for a least privilege policy to be "correct"?
  - Is it sufficient to match all operations in a code base?
  - Can you get a developer to document all intended accesses?

- Reality: least privilege is a non-verifiable security goal
  - We will discuss some verifiable goals later (e.g., IFC)

Information Flow

- Information can only flow in one direction
  - Towards more secret objects for confidentiality
  - Towards lower integrity objects for integrity
- Confidentiality
  - Processes cannot read objects that are "more secret"
  - In addition, processes cannot write data to objects that are "less secret" than they are
    - How does this prevent Trojan horse attacks?
- Integrity
  - Processes cannot write objects that are "higher integrity"
  - In addition, processes cannot read objects that are "lower integrity" than they are
    - How does this prevent Unexpected Attack Surfaces?

Information Flow

- Access control that focuses on information flow restricts the flow of information among subjects and objects
  - Regardless of functional requirements
- Confidentiality
  - Processes cannot read unauthorized secrets
  - Processes cannot leak their own secrets to unauthorized processes
    - Claim: Prevent Trojan horse attacks
- Integrity
  - Processes cannot write objects that are "higher integrity"
  - In addition, processes cannot read objects that are "lower integrity" than they are
    - Claim: Prevent attacks from Untrusted Inputs

Prevent Trojan Horses

- Information Flow Goal
  - Prevent Trojan horse attacks
- Intuition: Prevent flow of secrets to public subjects or objects

Information Flow

- Suppose O_1 must be secret to J_1 only
- No information flow from O_1 to either J_2 or J_3
  - What can you remove to protect the secrecy of O_1?

	O_1	O_2	O_3
J_1	R	RW	-
J_2	-	R	-
J_3	-	R	RW

Denning Security Model

- Information flow model $FM = (N, P, SC, x, y)$
  - N: Objects
  - P: Subjects
  - SC: Security Classes
  - x: Combination
  - y: Can-flow relation
- N and P are assigned security classes ("levels" or "labels")
- $SC_1 + SC_2$ determines the resultant security class when data of security classes SC_1 and SC_2 are combined
- $SC_2 \longrightarrow SC_1$ determines whether an information flow is authorized from security class SC_2 to SC_1
- SC, $+$, and $\longrightarrow$ define a lattice among security classes

Denning Security Model

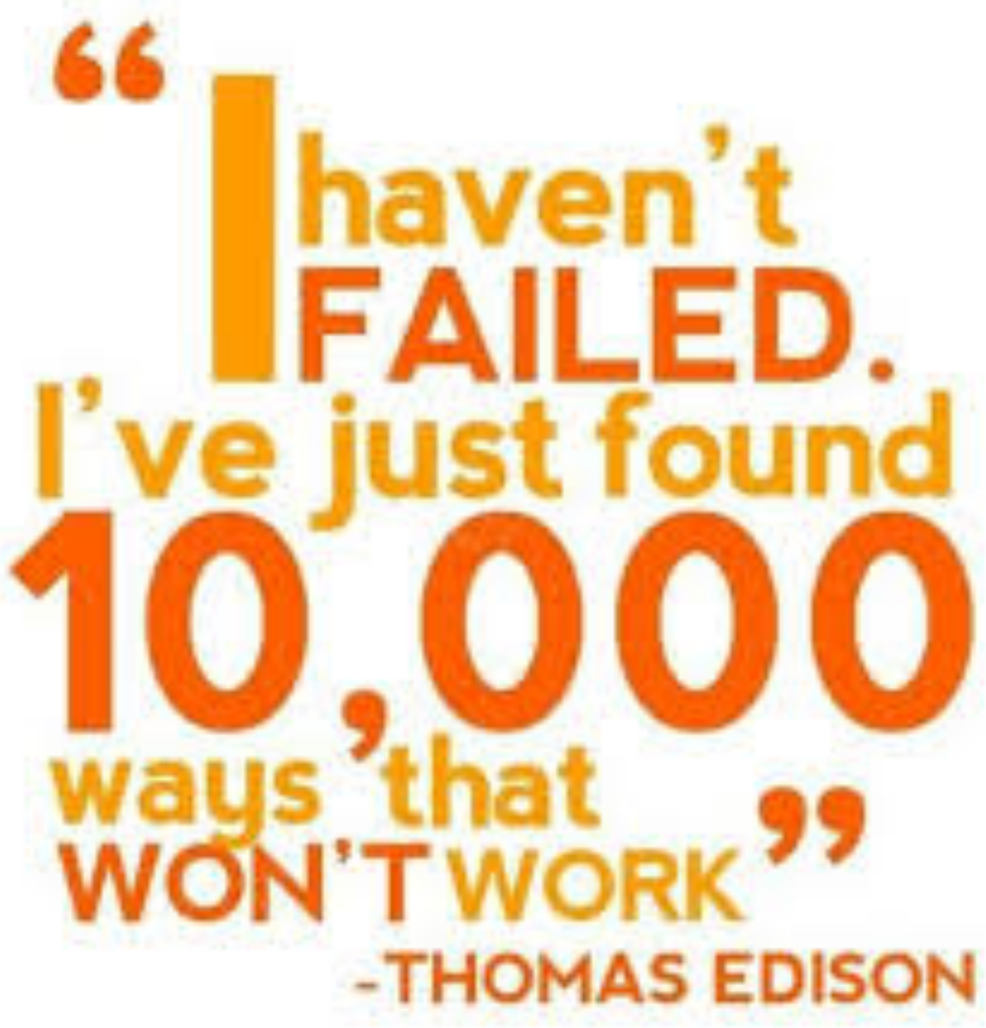
- Preventing Trojan horse attacks
  - Secret files are labeled SC_1 (secret)
  - Secret user logs in and runs processes that are labeled SC_1 (secret)
  - Public objects are labeled SC_2 (public)
  - Only flows within a class or from SC_2 to SC_1 are authorized (public to secret)
  - When data of SC_1 and SC_2 are combined, the resultant security class of the object is SC_1 (public and secret data make secret data)
- How does this prevent a Trojan horse from leaking data?

Information Flow

- Does information flow security impact functionality?

Information Flow

- Does information flow security impact functionality?
  - Yes, so need special processes to reclassify objects
    - Called guards, but are assumed to be part of TCB
      - "Require" formal assurance :-P

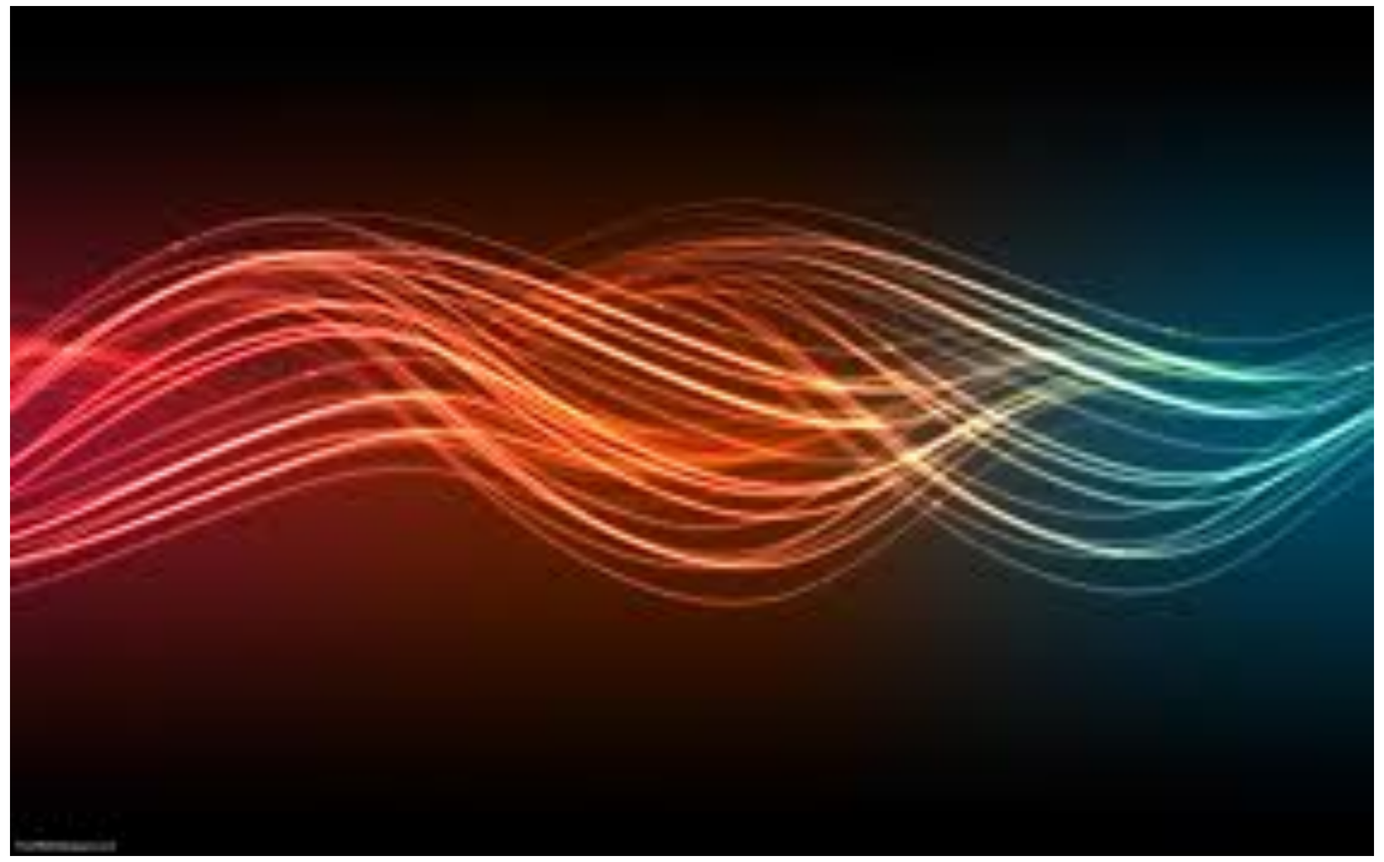


"I haven't FAILED. I've just found 10,000 ways that WON'T WORK"

-THOMAS EDISON

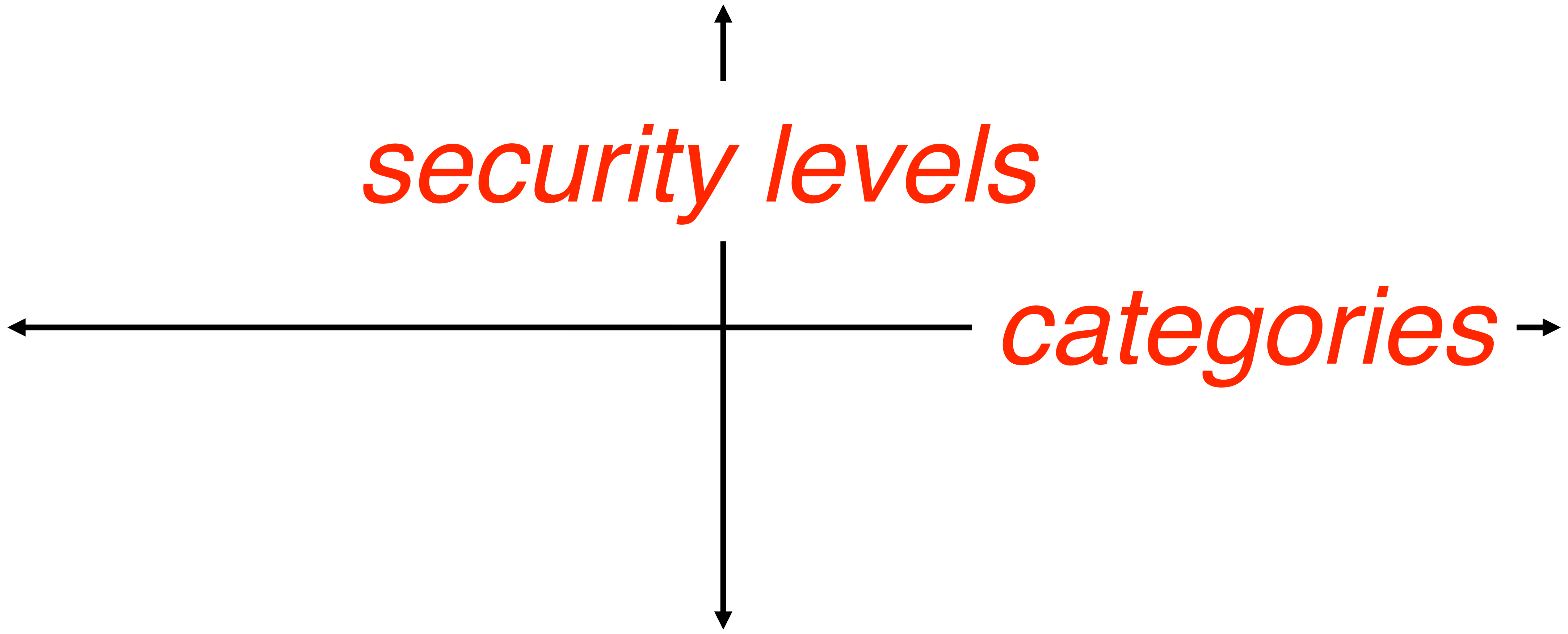
Information Flow Models

- Secrecy: Multilevel Security, Bell-La Padula
- Integrity: Biba, LOMAC



Multilevel Security

- A multi-level security system tags all objects and subjects with security tags classifying them in terms of sensitivity/access level.
  - We formulate an access control policy 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), 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

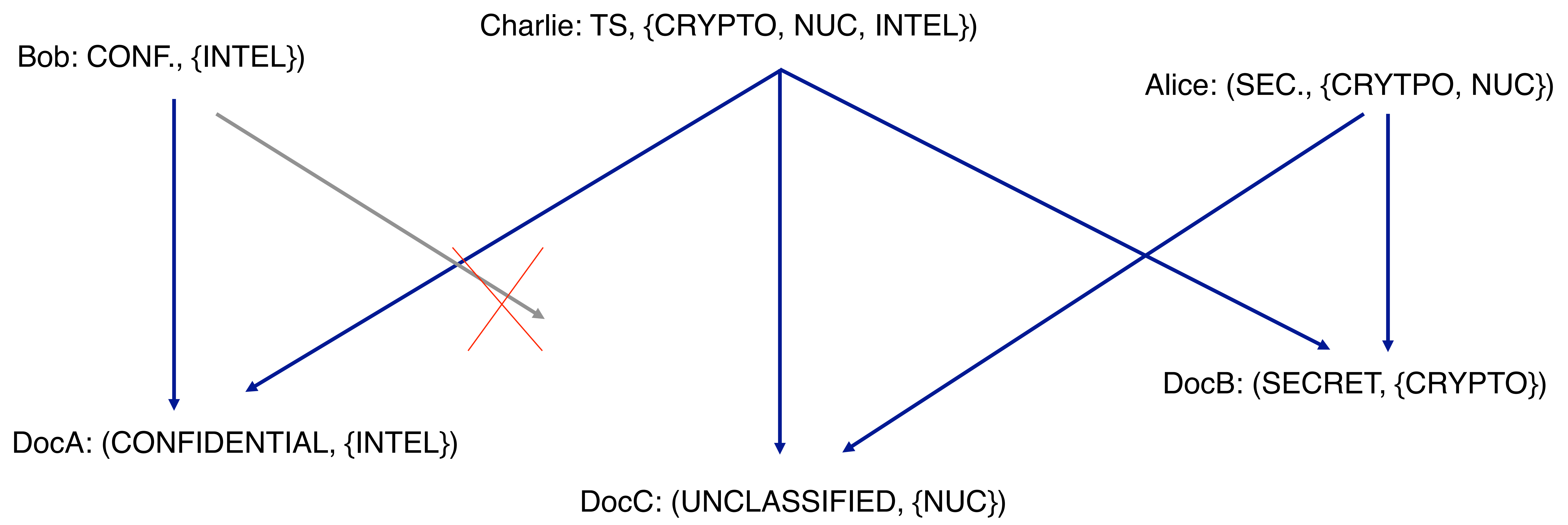
- All subjects are assigned clearance levels and compartments
    - Alice: (SECRET, {CRYTPO, NUC})
    - Bob: (CONFIDENTIAL, {INTEL})
    - Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL})

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

Multilevel Security

- Access is allowed if

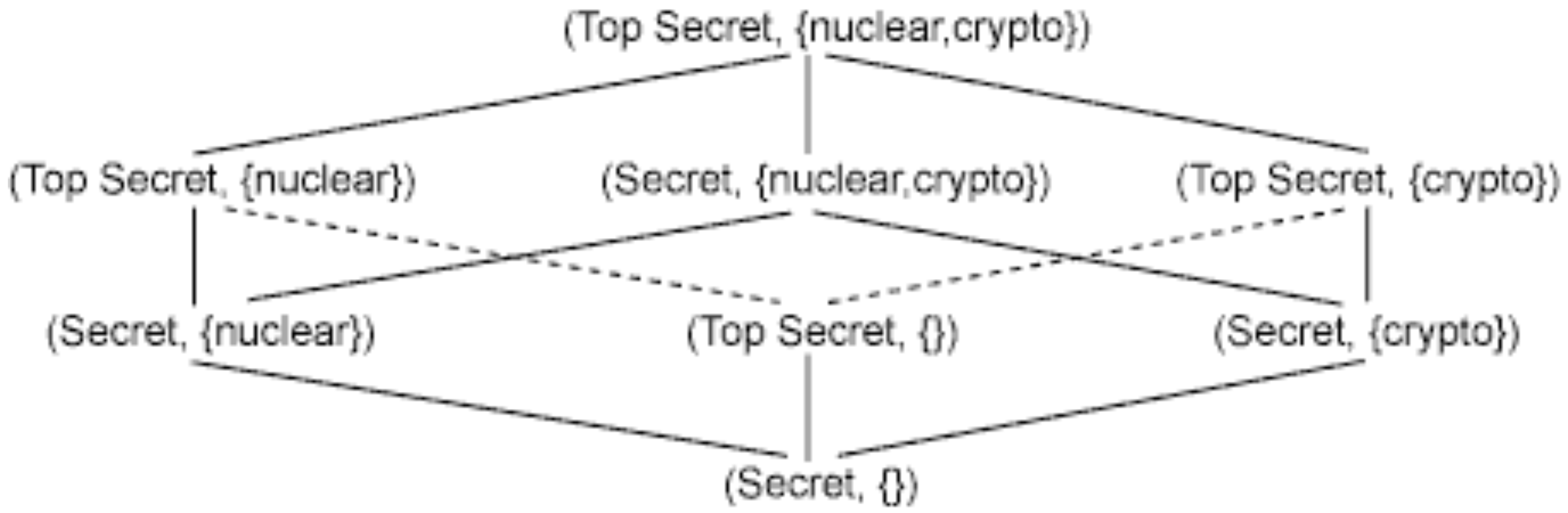
  subject clearance level >= object sensitivity level *and* subject categories ⊇ object categories (*read down*)



- Q: What would *write-up* be?

Bell-La Padula 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*".



Biba Model

- MLS as presented before talks about who can "read" a secret document (confidentiality)
- Integrity states who can "write" a sensitive document
  - Thus, who can affect 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
  - 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})

BLP vs. Biba

BLP is about secrecy

- Read: "no read up"
  sub level >= obj level and
  sub cat ⊇ obj cat
- Write: "no write down"
  obj level >= sub level and
  obj cat ⊇ sub cat

Biba is about integrity

- Read: "no read down"
  obj level >= sub level and
  obj cat ⊇ sub cat
- Write: "no write up"
  sub level >= obj level and
  sub cat ⊇ obj cat

Window Vista Integrity

- Integrity protection for writing
- Defines a series of protection level of increasing protection
  - installer (highest)
  - system
  - high (admin)
  - medium (user)
  - low (Internet)
  - untrusted (lowest)
- Semantics: If subject's (process's) integrity level dominates the object's integrity level, then the write is allowed

Vista Integrity

S1(installer)

O1(admin)

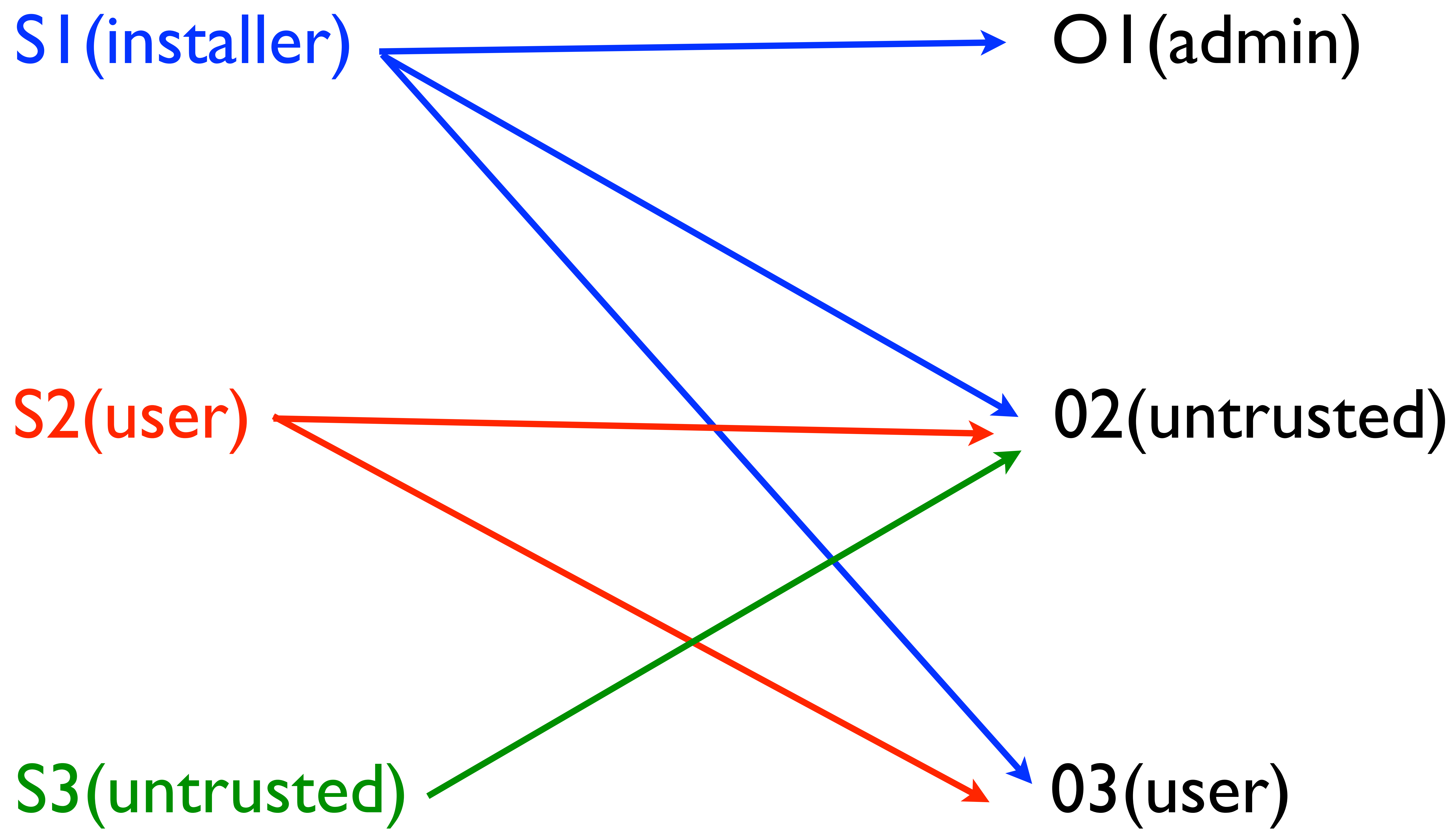
S2(user)

02(untrusted)

S3(untrusted)

03(user)

Vista Integrity



S1(installer)

O1(admin)

S2(user)

02(untrusted)

S3(untrusted)

03(user)

Reduce Integrity Restrictiveness

- Can we allow processes to read lower integrity data without compromising information flow?
  - Still don’t trust the process to handle lower integrity inputs without being compromised
- Insight: Could change the integrity level of each process based on the data it accesses

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

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.

Take Away

- Claim: Traditional access control approaches (UNIX and Windows) do not enforce security against a determined adversary
    - (1) Trojan horses and confused deputies violate security goals
    - (2) DAC models prevent goals from being enforced
- Mandatory Access Control (MAC) is the way these can be achieved
    - MAC policies
        - Information flow models (MLS, Biba)
        - Least privilege MAC is often used (see SELinux)