DSCI 519: Foundations and Policy for Information Security

Formal Security Policy Models

Tatyana Ryutov

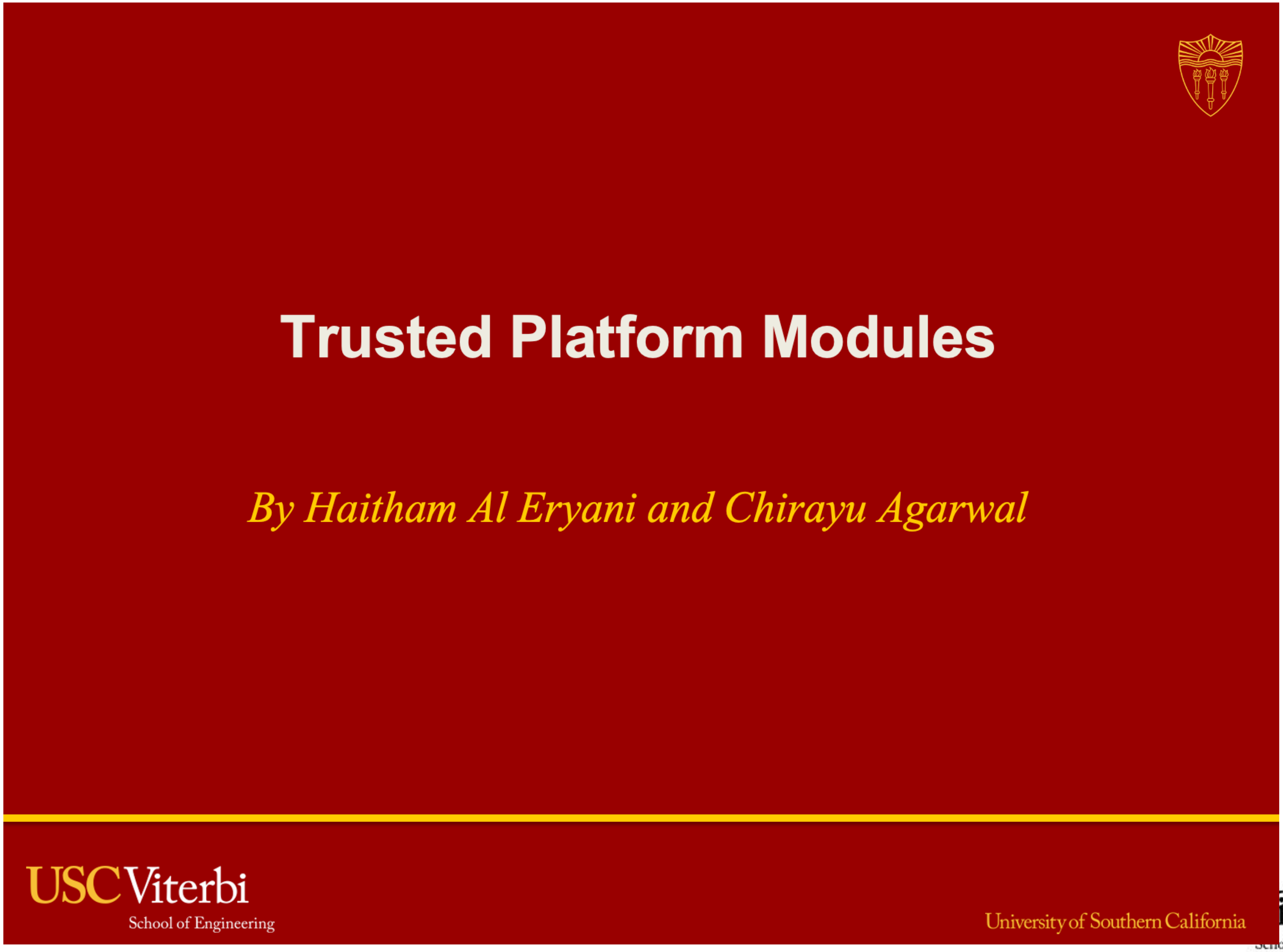
Outline

- Review
- Interpreting Reference Monitor components
- Motivation for formal security policy model
- Examples of policy models:
  - Access Control Matrix
  - Information flow models
  - State machine models, basic security theorem
- Bell-LaPadula (BLP) model
  - Preliminary concepts: inductive reasoning, MAC access classes, lattice structures

Reminder

- Quiz 1 is posted on D2L
- You will have 45 minutes to answer 10 questions
- Complete by Sunday, September 11th, 11:59pm
- Covers lectures 1-2

Presentation 1



Trusted Platform Modules

By Haitham Al Eryani and Chirayu Agarwal

USC Viterbi
School of Engineering

University of Southern California

L3.Q6

!

- Think about all the material covered today
- Indicate two specific topics/questions for review next lecture
  - focused questions/topics I can give you an answer to
- If everything is clear, it’s OK to say: “none”

USC Viterbi
School of Engineering

Your Questions

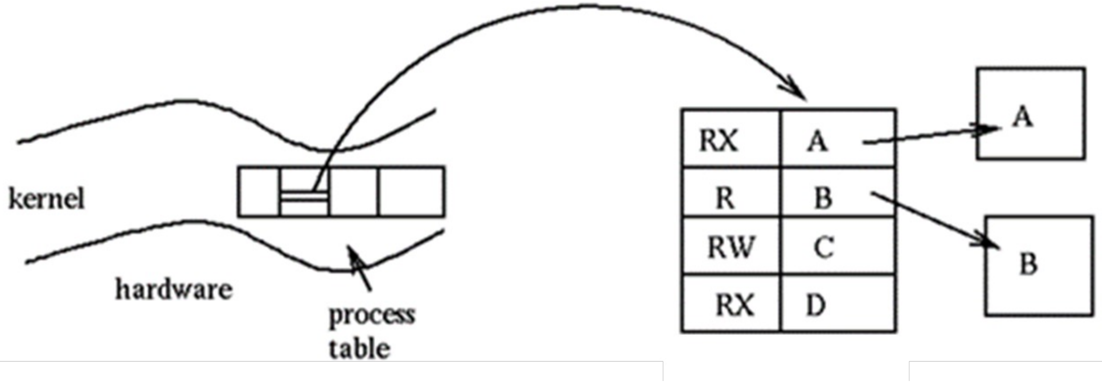
- How much specific information should we know about the different Operating Systems, such as Windows, MacOS, Linux, etc.? What Operating System is the safest to use? Is there an easy way to make all of them trusted?
- Do distributed architecture have single RM or multiple implementations?
- I was a little confused on dedicated system mode, I am not sure how important that subject is but maybe go over that a little more.
- Can we please review the MAC Modes of Operation, specifically Cross-Domain Solution
- Can you review the capability list?
- Did not fully understand the last part of the lecture covering RM (reference machines) probably because I was getting mentally tired. A refresher next class would be helpful.

Review: C-list

- Capability Lists (C-list): does not include identity for authorization system to check
  - It doesn't matter who presents the capability
  - With ACLs we assume that authentication is unforgeable
  - With capabilities, we need to make capabilities unforgeable
- Implementation example:
  - Only provide processes with **indices** to the table
    - A process can use only these indices, and thus a process can only talk about capabilities that it has, making it impossible to create new capabilities

Objects	Object1	Object 2	Object 4
	read	read, write	write

Access Capability List for Bob process

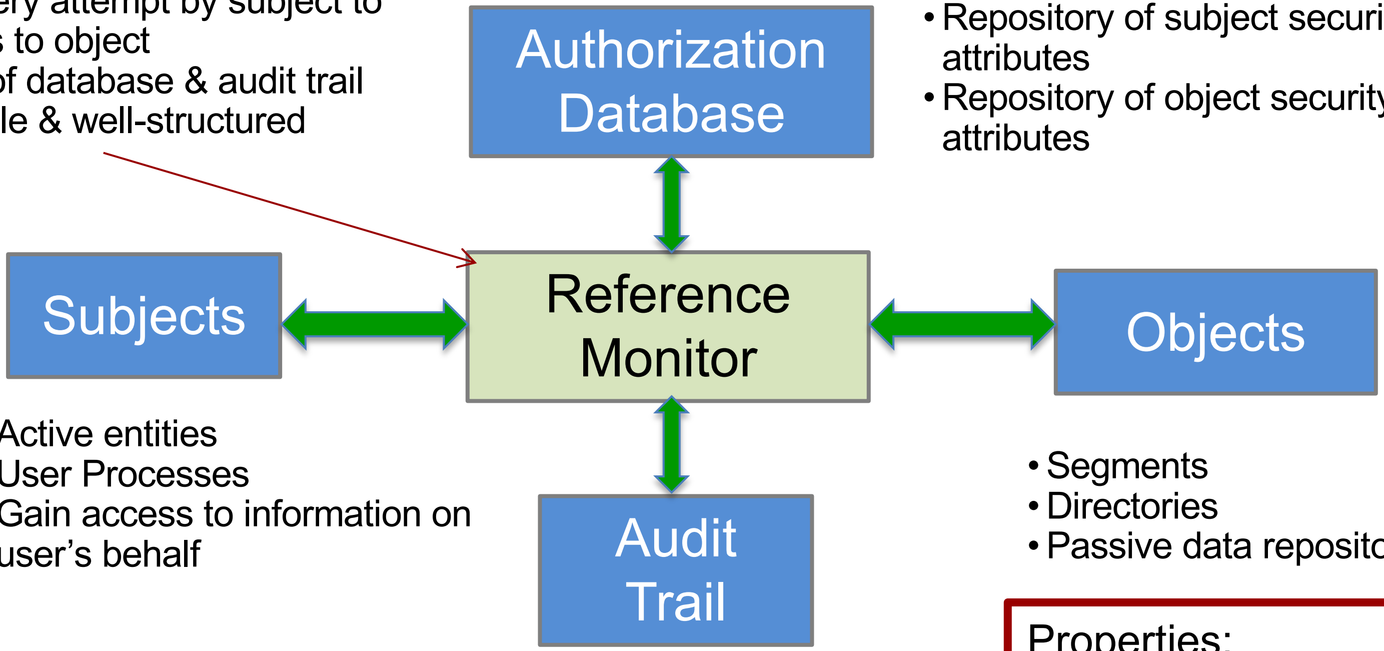


RX	A
R	B
RW	C
RX	D

Review: Reference Monitor Abstraction

- Conceptual access control abstraction
- RM does not specify a particular access control policy

- Enforces security policy
- Mediate every attempt by subject to gain access to object
- Tamperproof database & audit trail
- Small, simple & well-structured



- Repository of subject security attributes
- Repository of object security attributes

- Active entities
- User Processes
- Gain access to information on user's behalf

- Segments
- Directories
- Passive data repositories

- Record of all security-related events

Properties:

1. Tamperproof
2. Non-bypassable
3. Verifiable

Please review the verifiable principle of the Reference Monitor. does it essentially mean that the isolation and non-bypassible is verified because the programming is simple and straight forward enough to verify?

Does verifiable mean able to manually review the code, and if so, does that mean all COTS software and hardware should never be trusted?

USC Viterbi
School of Engineering

Reference Monitor in Computer System

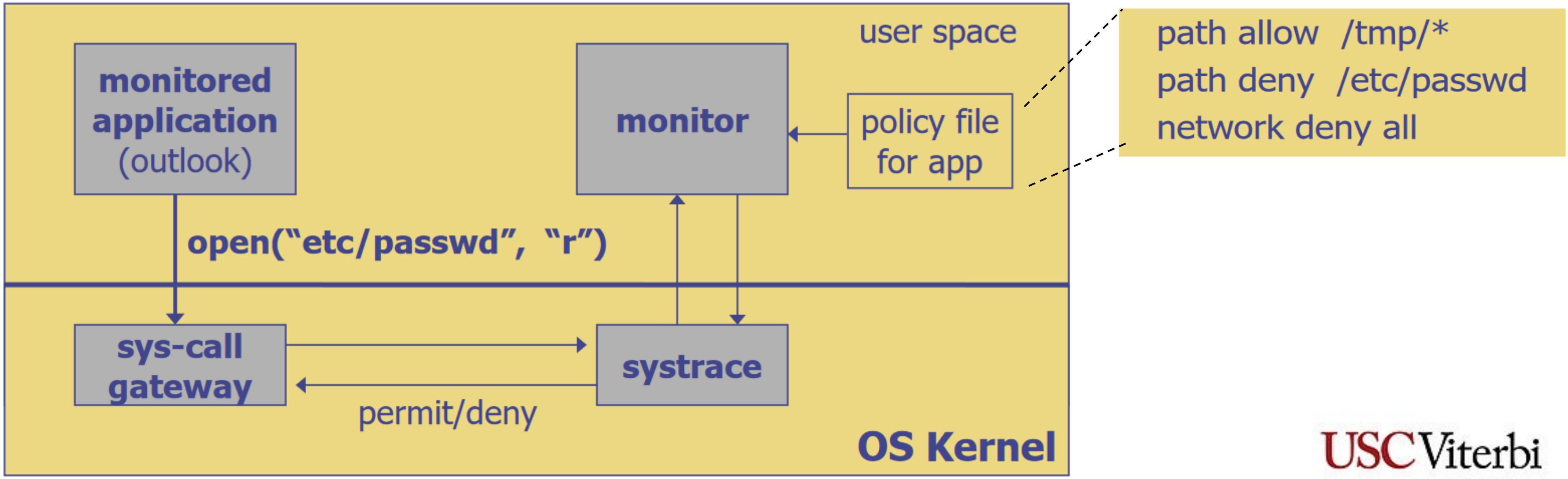
- Most primitive portion relied on to control access
- Reference Validation Mechanism (RVM)
  - Combination of hardware & software that implements RM
  - Defined as **security kernel** (TCSEC – Orange Book)
- Use Reference Monitor to show system “secure”
  - I.e., verify that computer system meets policy
  - Access control implementation corresponds to policy
- Verifying policy based on interpretation choices
  - Choices for the “subjects” of reference monitor (RM)
  - Choices for “objects” for the RM
  - Choices for formulation of “access control policy”

Security Kernel: Example

- Security kernel - primitive operating system
  - Responsible for enforcing security policy for entire OS
  - Implements reference monitor
- Example: Gemini Corporation's Standard Operating Systems (GEMSOS) Security Kernel
  - Mandatory security reference monitor
  - Uses BLP for MAC confidentiality and for MAC integrity
  - Runs on the available commercial hardware, the Intel x86 architecture
  - Implemented in Pascal, ≈ 12K lines of code
- Evaluated as A1 (highest assurance system)
  - http://www.aesec.com/eval/NCSC-FER-94-008.pdf

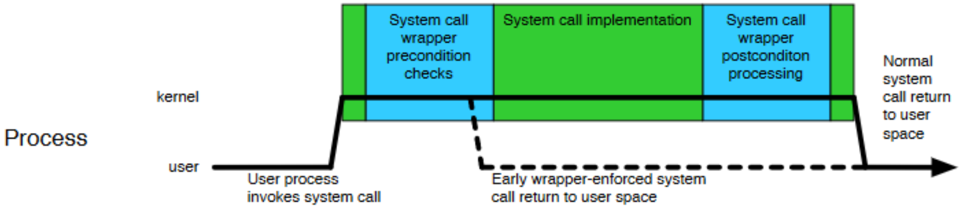
Example: System Call Interposition (SCI)

- Observation: to subvert host system app must make system calls
  - To delete/overwrite files: unlink, open, write
  - To do network attacks: socket, bind, connect, send
- Solution: monitor app system calls and block unauthorized calls
- Examples:
  - Systrace, ships with OpenBSD
  - Google Native Client (NaCi), shipped with Google Chrome



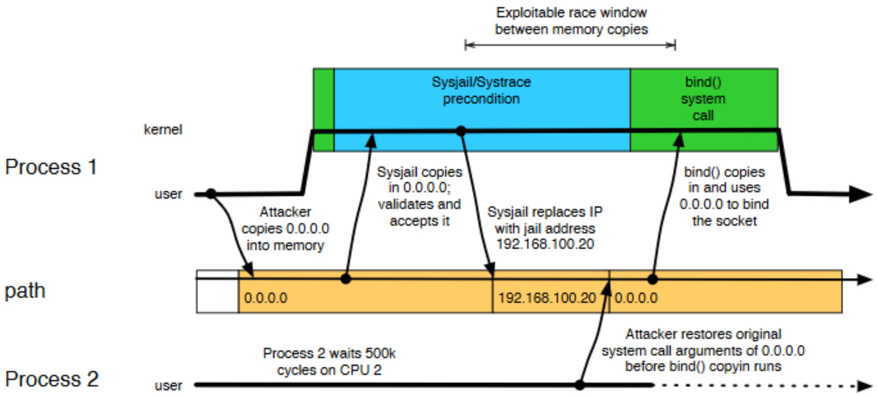
Example: Bypassing SCI

- Time Of Use Time Of Check (TOCTOU): race condition between checking and using parameter
  - Access control checks are non-atomic with the operations they protect
- System-call wrapper races can lead to partial or complete bypass of access control and audit


System call wrapper precondition checks
System call implementation
System call wrapper postconditon processing
kernel
Process
user
Normal system call return to user space
User process invokes system call
Early wrapper-enforced system call return to user space

time

```
if(access("foo")) {
     context switch              symlink("secret", "foo")
  fd = open("foo");
  read(fd,…);
  …
}
```


Exploitable race window between memory copies
Sysjail/Systrace precondition
bind() system call
kernel
Process 1
user
Attacker copies 0.0.0.0 into memory
Sysjail copies in 0.0.0.0; validates and accepts it
Sysjail replaces IP with jail address 192.168.100.20
bind() copies in and uses 0.0.0.0 to bind the socket
path
0.0.0.0
192.168.100.20
0.0.0.0
Process 2 waits 500k cycles on CPU 2
Attacker restores original system call arguments of 0.0.0.0 before bind() copyin runs
Process 2
user

SCI was developed to avoid OS modification: add on approach
These vulnerabilities derive from the fundamental architectural separation of the wrapper from native kernel synchronization strategies

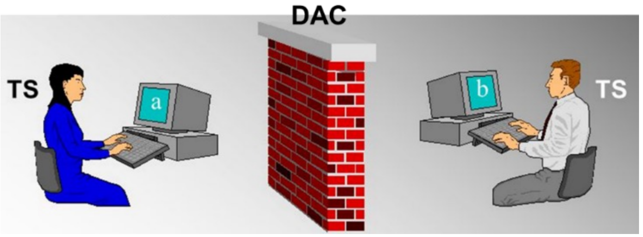
Outline

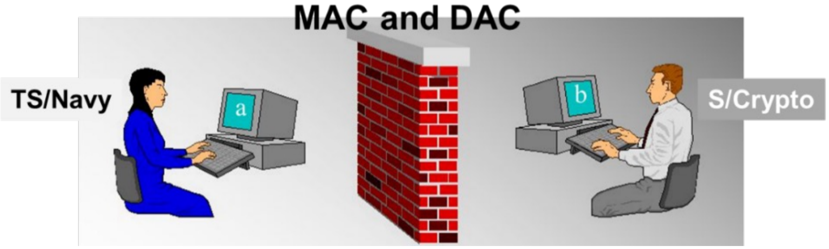
- Review
- Interpreting Reference Monitor components
- Motivation for formal security policy model
- Examples of policy models:
  - Access Control Matrix
  - Information flow models
  - State machine models, basic security theorem
- Bell-LaPadula (BLP) model
  - Preliminary concepts: inductive reasoning, MAC access classes, lattice structures

Interpret Modes of Operation

- Dedicated
  - User has clearance for all information (no MAC)
  - User has need-to-know for all information (no DAC)
- System High
  - User has clearance for all information (no MAC)
  - User has need-to-know for some information
    - System enforced DAC need-to-know
- Multilevel
  - System enforces MAC and DAC
- Goal: interpretation for each mode of operation

What does air gap mean? Why System High Mode is DAC, not MAC?





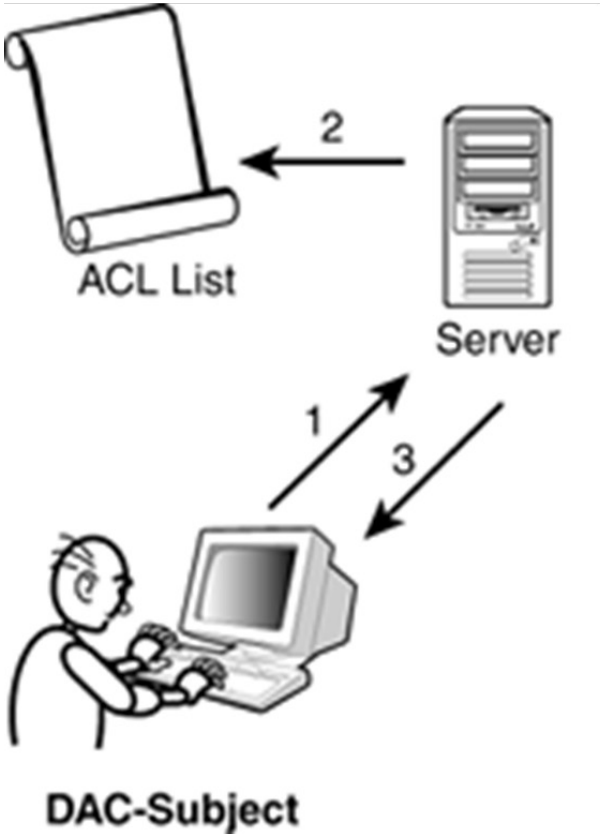
We have dedicated mode, System high mode and Multilevel mode for and MAC, DAC and RM. Does these modes apply to other policies?

Interpret "Dedicated" Mode for RM

- Consider a computer system
- Interpretations for "dedicated" mode of operation
  - Subjects – every user has total access
    - All processes in computer (it is the active entities)
  - Objects
    - Collection of all the information in computer (passive)
  - Authorization database
    - "Policy" is all subjects authorized access to all objects
  - Audit file
    - External record of **physical** access, for identified user

Interpret "System High" Mode for RM

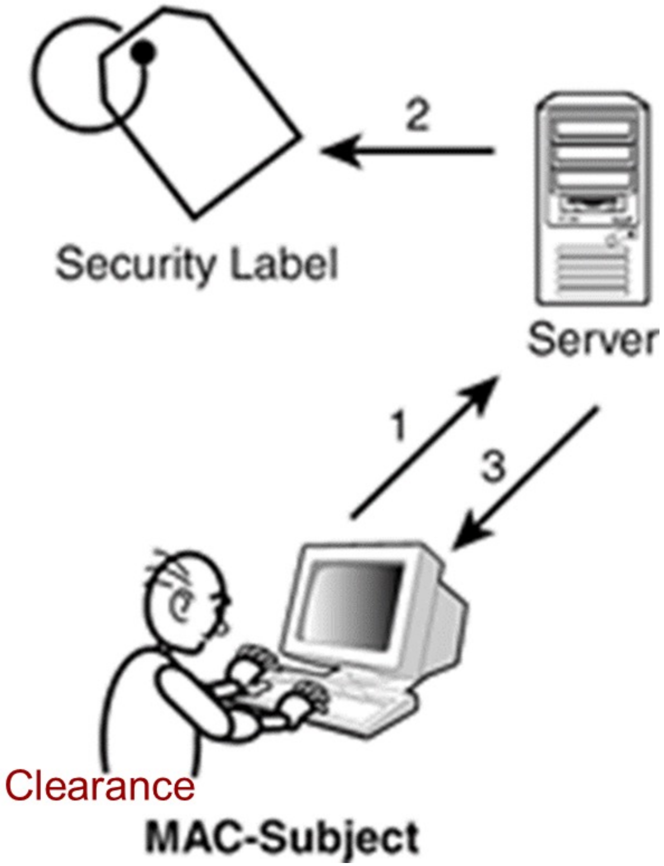
- Interpret "system high" mode of operation
  - Subjects
    - Each process for identified user
    - Surrogate for user who is subject to DAC policy
  - Objects
    - Distinct repositories or information containers, e.g., files
  - Authorization database
    - DAC authorizations for users and groups of users
  - Audit file
    - Can record object access requests, with identity of user



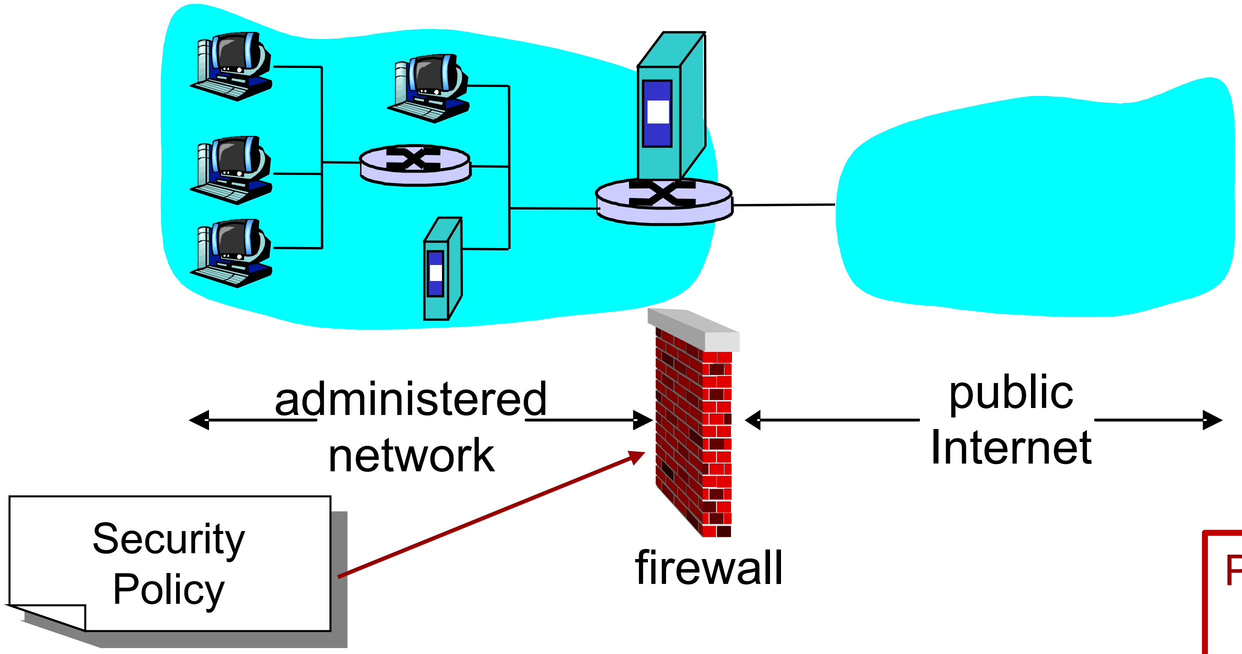
In class there is an example that DAC-Subject access ACL list, is it possible that DAC-Subject access. Why DAC is binded with ACL but not Capability Lists?

Interpret "Multilevel" Mode for RM

- Interpretations for "multilevel" mode of operation
  - Subjects
    - Processes for users, each with a label for clearance
  - Objects
    - Information repository, with MAC security label, e.g., segment
  - Authorization database
    - DAC authorizations for identified users and groups
    - MAC authorization - clearance needed for classification
  - Audit file
    - Record object access request, with user identity & labels



Example: Firewall as RM



Properties:

1. Tamperproof
2. Non-bypassable
3. Verifiable

- Check Point Security vulnerabilities:
  - https://www.cvedetails.com/vulnerability-list/vendor_id-136/Checkpoint.html
- CISCO security vulnerabilities
  - https://us-cert.cisa.gov/ncas/current-activity/2020/03/05/cisco-releases-security-updates

Properties of RM Abstraction

- Only known high assurance cyber security basis
  - Provides the basic theory of computer security
  - Essential to trustworthy response to subversion threat
    - I.e., (isolation) tamper-proof
- Also used to assess low to medium assurance
  - E.g., RM functions in an application program
  - Fails to meet three RM principles
  - But still useful way of looking at the system

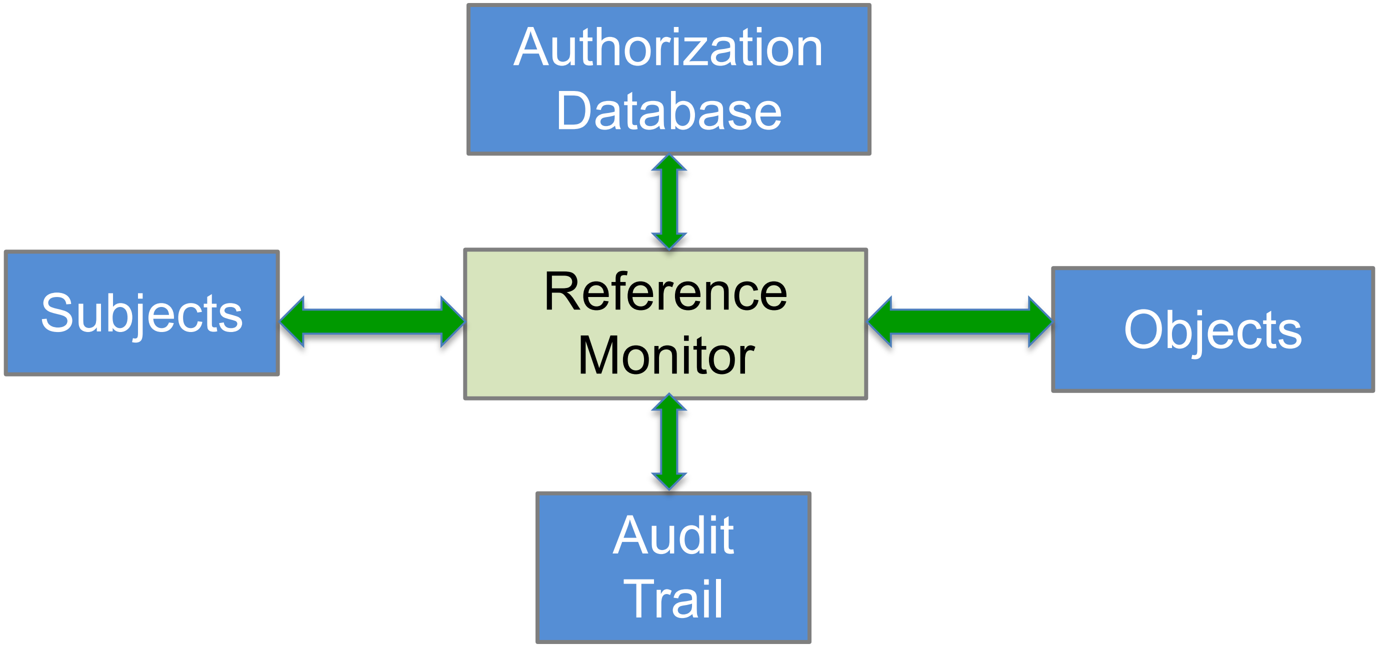
RM Open Issues

- Complete mediation requires that all security-sensitive operations are identified
  - Often these operations are not precisely defined
- Tamperproof
  - Most systems have a trusted computing base that is too large to determine whether tampering is prevented
- Verification is the most difficult to satisfy
  - Requires designing a general algorithm to prove that an arbitrary program behaves correctly
    - Equivalent to solving the Halting problem

Outline

- Review
- Interpreting Reference Monitor components
- Motivation for formal security policy model
- Examples of policy models:
  - Access Control Matrix
  - Information flow models
  - State machine models, basic security theorem
- Bell-LaPadula (BLP) model
  - Preliminary concepts: inductive reasoning, MAC access classes, lattice structures

RM Abstraction: Verifiability



Properties:

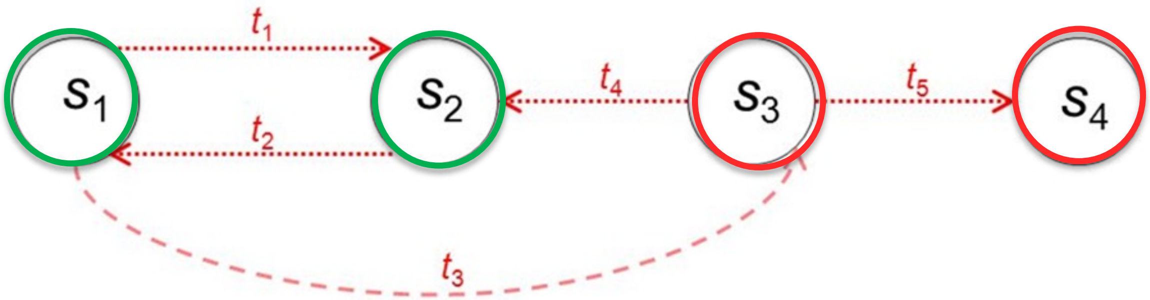
1. Tamperproof
2. Non-bypassable
3. Verifiable

Formal Security Policy Model (FSPM)

- Provides a precise definition of the security policies
  - defines the policy rules and characteristics
  - defeats ambiguity (e.g., “An applet should connect to the same server from which it originated”)
- View as formalization and particularization of RM
- Simple & abstract
  - necessary to verification process
- Must include a proven security theorem
  - establishes that the model’s behavior always complies with the security requirements

Key Concepts from Readings

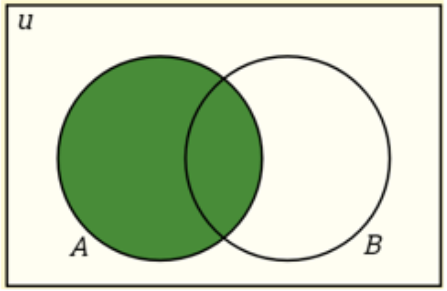
- A security policy **partitions** the states of the system into a set of authorized (secure) states and a set of unauthorized (non-secure) states
- A security mechanism **prevents** the system from entering a "non-secure" state



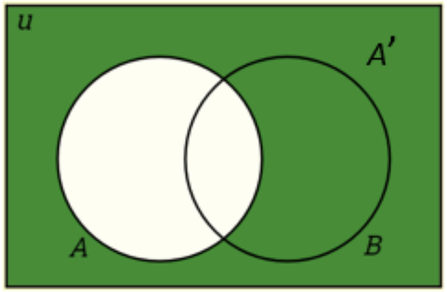
Breach of security

S_1 and S_2 - authorized states, S_3 and S_4 – unauthorized states

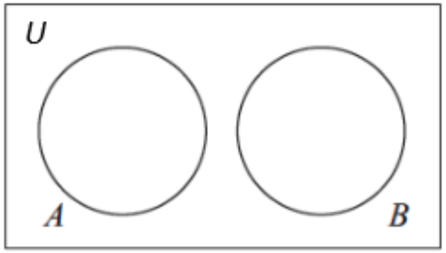
Recall: Set Operations and Venn Diagrams



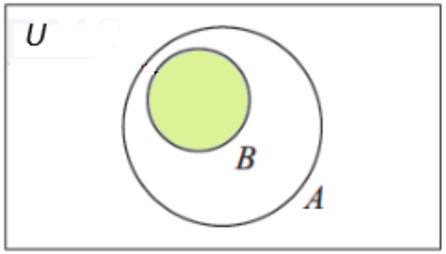
Set *A*



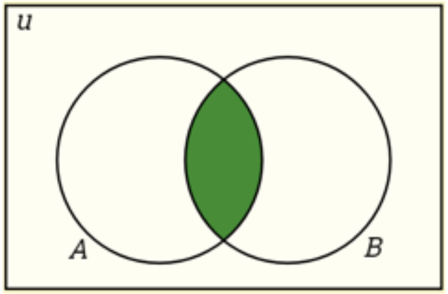
A’ the complement of *A*



A and *B* are disjoint sets

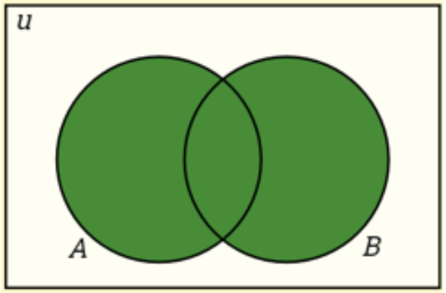


B is proper subset of *A* $B \subset A$



Both A and B $A \cap B$

A intersect *B*



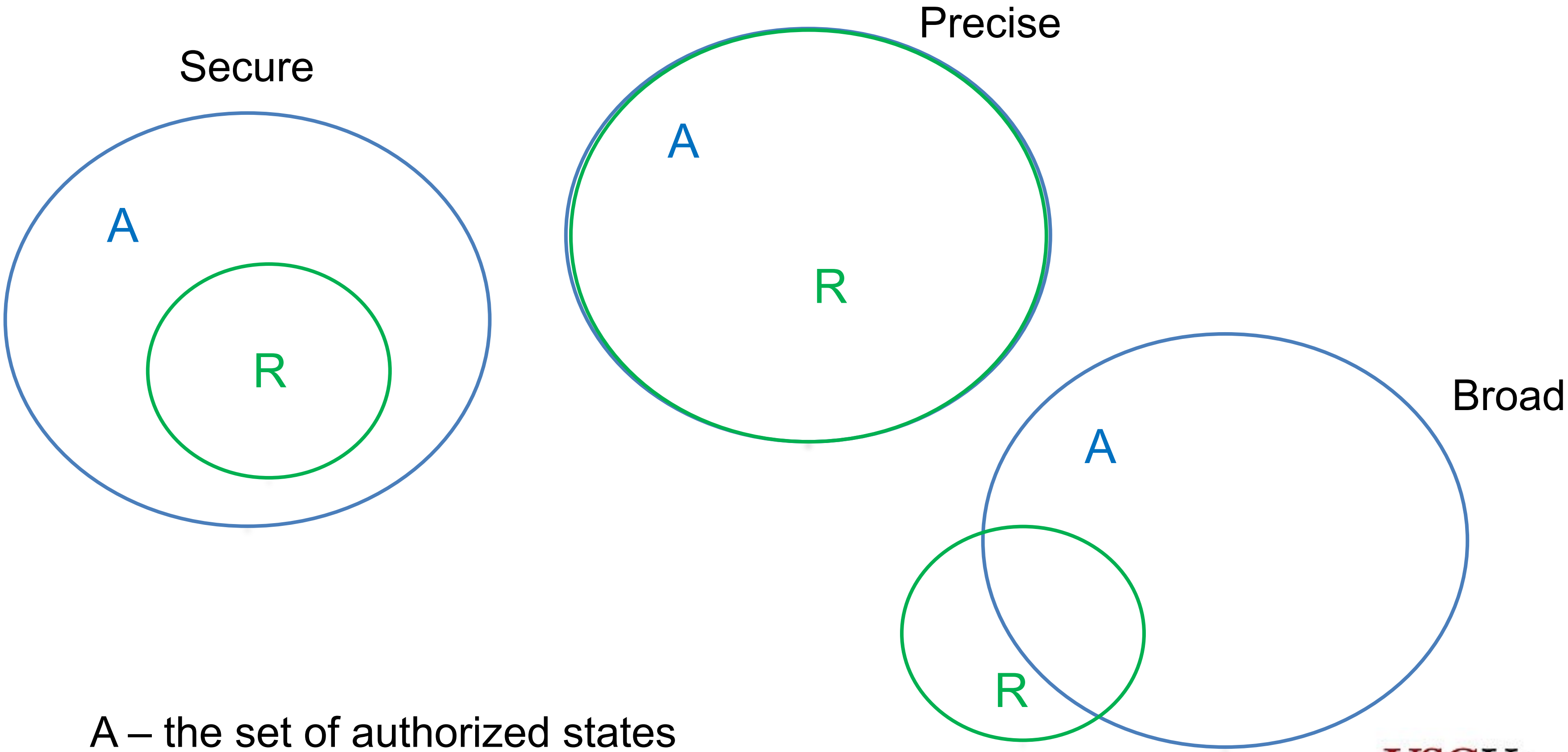
Either A or B $A \cup B$

A union *B*

USC Viterbi School of Engineering

Types of Security Mechanisms

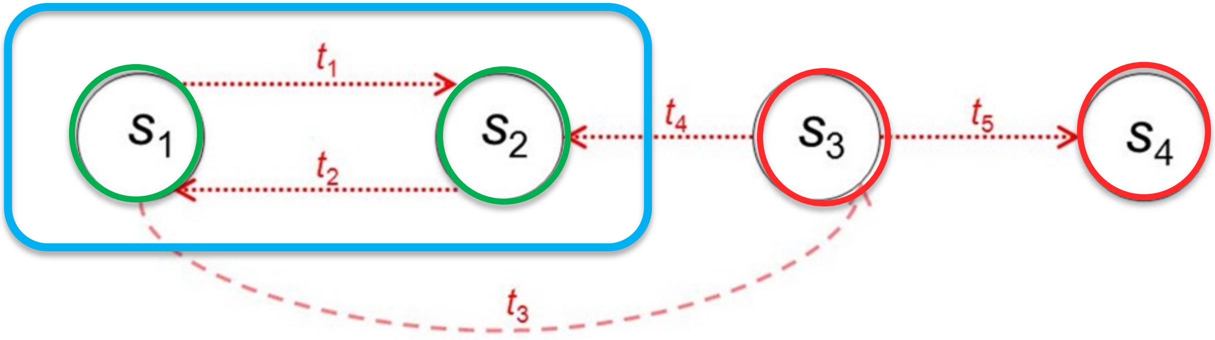
- Security policy can be enforced by security mechanisms that are either secure, precise, or broad [Bishop]



A – the set of authorized states
R – the set of reachable states

FSPM is Linchpin for Verification

- Is **linchpin** for verification that system is secure
- Translate non-computable negative requirements
  - E.g., “unauthorized access always prohibited”
- Create sufficient verifiable positive requirements
  - E.g., “system always in a secure state”
- Basis to meet RM “verifiability” principle
  - Compare to “patch Tuesday” model

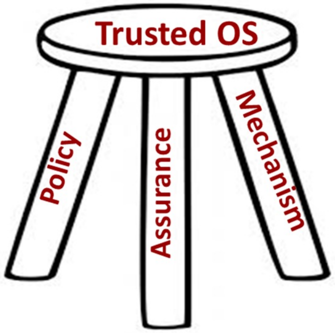


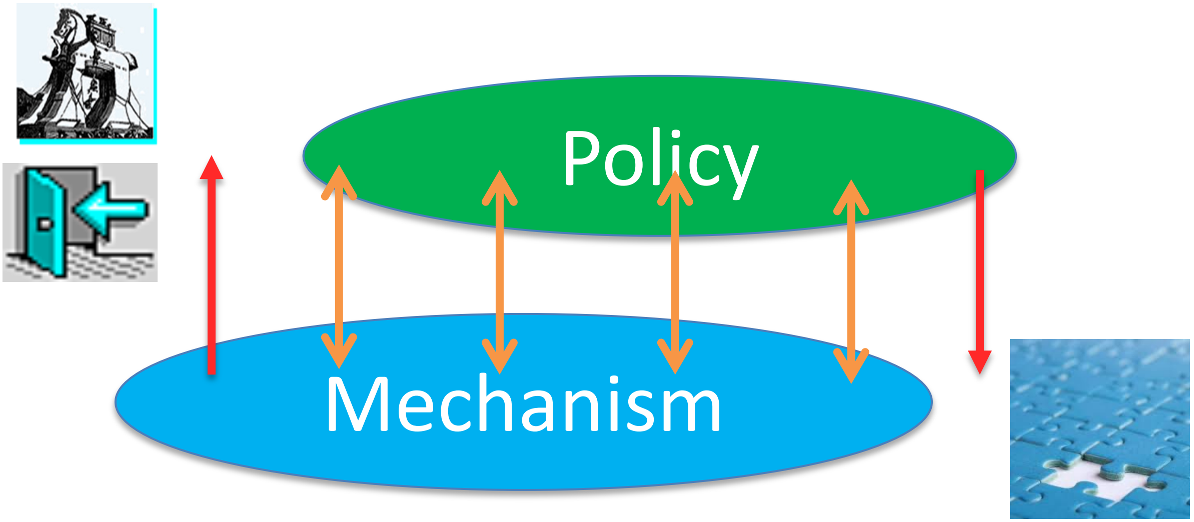
Breach of security

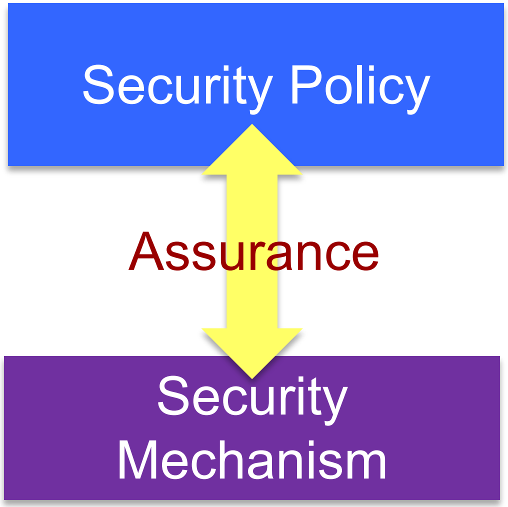
S_1 and S_2 - authorized states, S_3 and S_4 – unauthorized states

Recall: Mechanisms vs. Policy

- Protection measures must be traceable to policy
- Policy must be traceable to the measures
- If traceability fails, usually something breaks
  - Information is not adequately protected
  - Implementation contains superfluous components






Security Policy

Assurance

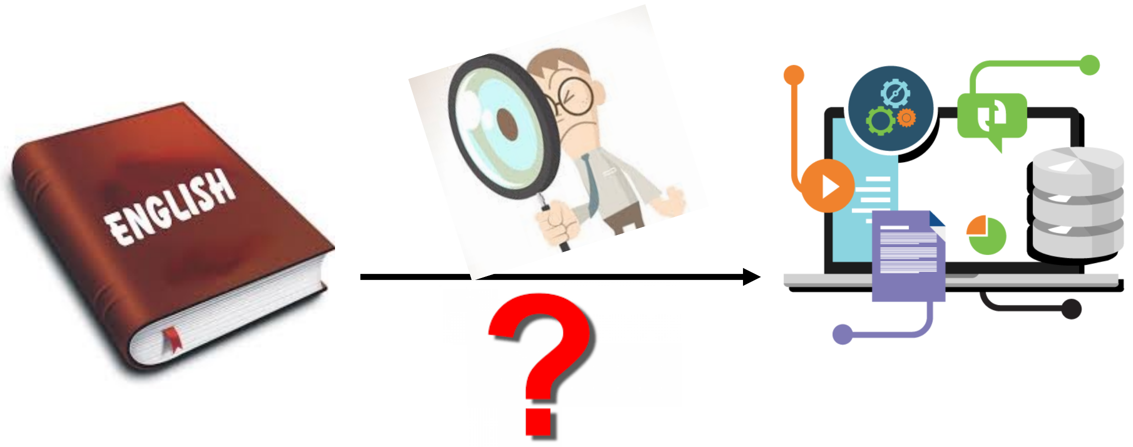
Security Mechanism

©Tatyana Ryutov

USC Viterbi
School of Engineering

Why FSPM?

- Enables leap from policy to software and HW (trusted computing base - TCB)
  - Provides an inspectable intermediate step
  - Formal description of the functions that the TCB will perform
- Security policy → FSMP → RM Implementation



Formalization of Reference Monitor

- Formalization and particularization of RM
  - Formalization in mathematical way
- Bridges from people-oriented world of security policy to computer-oriented RM subjects & objects
- Intended for verifying computer enforcement of policy
- Not easy for real systems

Outline

- Review
- Interpreting Reference Monitor components
- Motivation for formal security policy model
- Examples of policy models:
  - Access Control Matrix
  - Information flow models
  - State machine models, basic security theorem
- Bell-LaPadula (BLP) model
  - Preliminary concepts: inductive reasoning, MAC access classes, lattice structures

FSPM Abstraction for DAC Policy: Lampson’s Access Matrix Model

- Recording the protection state of the system
- Policy has discretionary access control (DAC)
- Represents protection state of the system
- Access matrix M can be abstraction for DAC
  - Subject S **read** access to object O ⇒ m [S, O] = read
  - Subject S **write** access to object O ⇒ m [S, O] = write

Access Matrix

Objects Subjects	**Object 1**	**Object 2**	**Object 3**	**Object 4**
Bob Process	read	read, write		write
Flo Process	read, write, **own**	write		
Alice Process	read	read	read	read, write
Dan Process	read		read, write, **own**	read

Information Flow Models

- Focus on the **flow** of information instead of on **individual accesses** to objects
  - Basis of design for the Bell-LaPadula model
- The goal of the information flow model is to prevent unauthorized information flow in any direction
- Exist in the context of programming languages
  - Can be applied to the variables in a program directly
    - Example: transfer of information from a variable **x** to a variable **y** in a given process
      - explicit information flow: y := x
      - implicit information flow: IF x == 0 THEN y :=1

Example: Tainted Flow Analysis

- The root cause of many attacks is trusting unvalidated input
  - Input from the user is tainted
  - Various data is used, assuming it is untainted
- Examples expecting untainted data
  - source string of **strcpy** (≤ target buffer size)
  - form field used in constructed SQL query (contains no SQL commands)
  - format string of **printf** (contains no format specifiers)
- Analysis problem: no tainted data flows:
  - for all possible inputs, prove tainted data will never be used where untainted data is expected
    - untainted annotation: indicates a trusted sink
    - tainted annotation: an untrusted source
    - no annotation means: not sure (analysis figures it out)
  - If flow analysis finds that tainted data could be used where untainted data is expected → potential security vulnerability
- A solution requires inferring flows in the program
  - What sources can reach what sinks
  - If any flows are illegal, i.e., whether a tainted source may flow to an untainted sink

Example: Format String Attack

- Adversary-controlled format string

```
char *name = fgets(…, network_fd);
printf(name);
```

  - Attacker sets name = `“%s%s%s”` to crash program
  - Attacker sets name = `“…%n…”` to write to memory
  - Yields code injection exploits

- Specify our requirement as a type qualifier

```
int printf(untainted char *fmt, …);
tainted char *fgets(…);
```

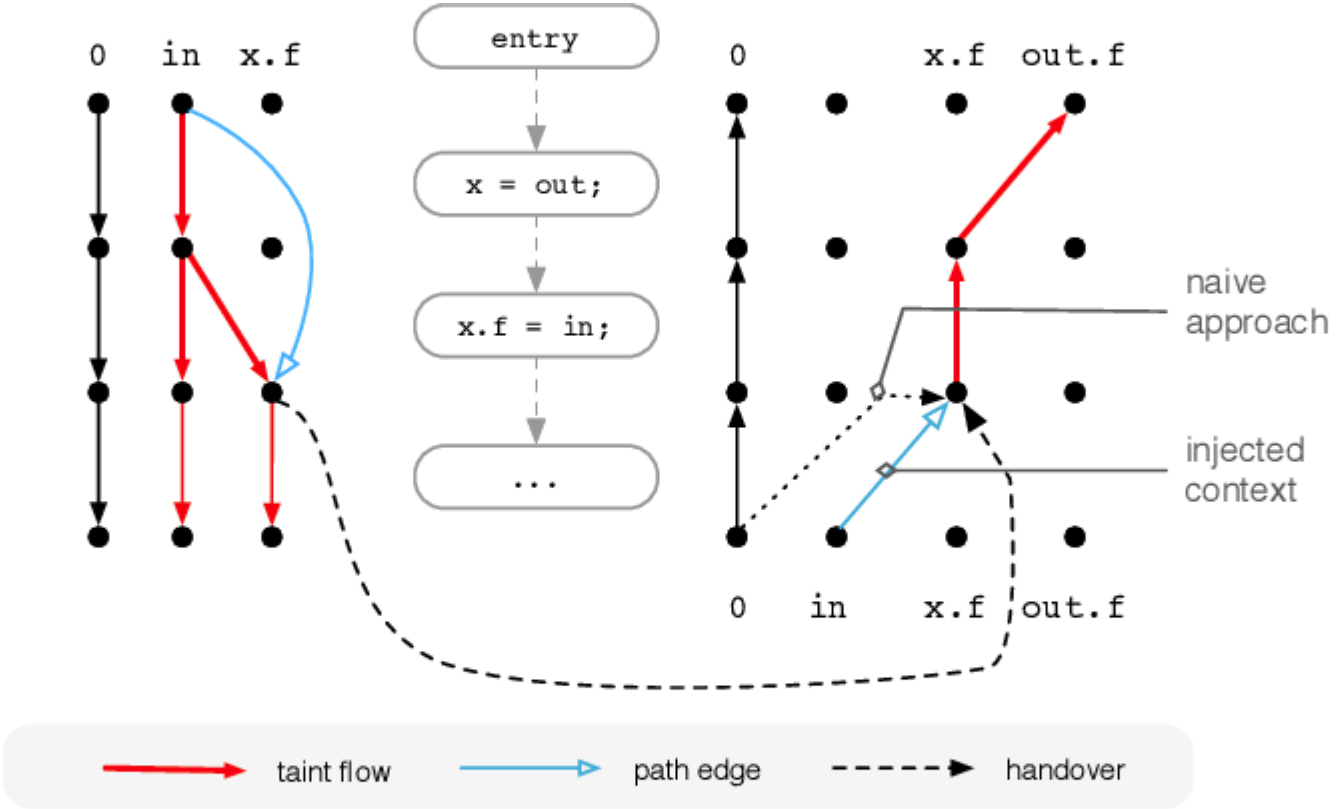
  – **tainted** = possibly controlled by adversary
  – **untainted** = must not be controlled by adversary

```
tainted char *name = fgets(…,network_fd);
printf(name); // FAIL: tainted ≠ untainted
```

USC Viterbi School of Engineering

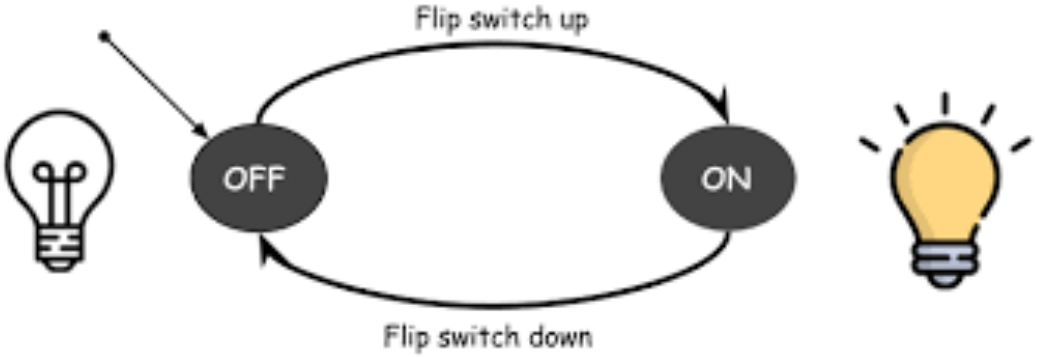
Taint Analysis Tool Example: FlowDroid

- https://github.com/secure-software-engineering/FlowDroid

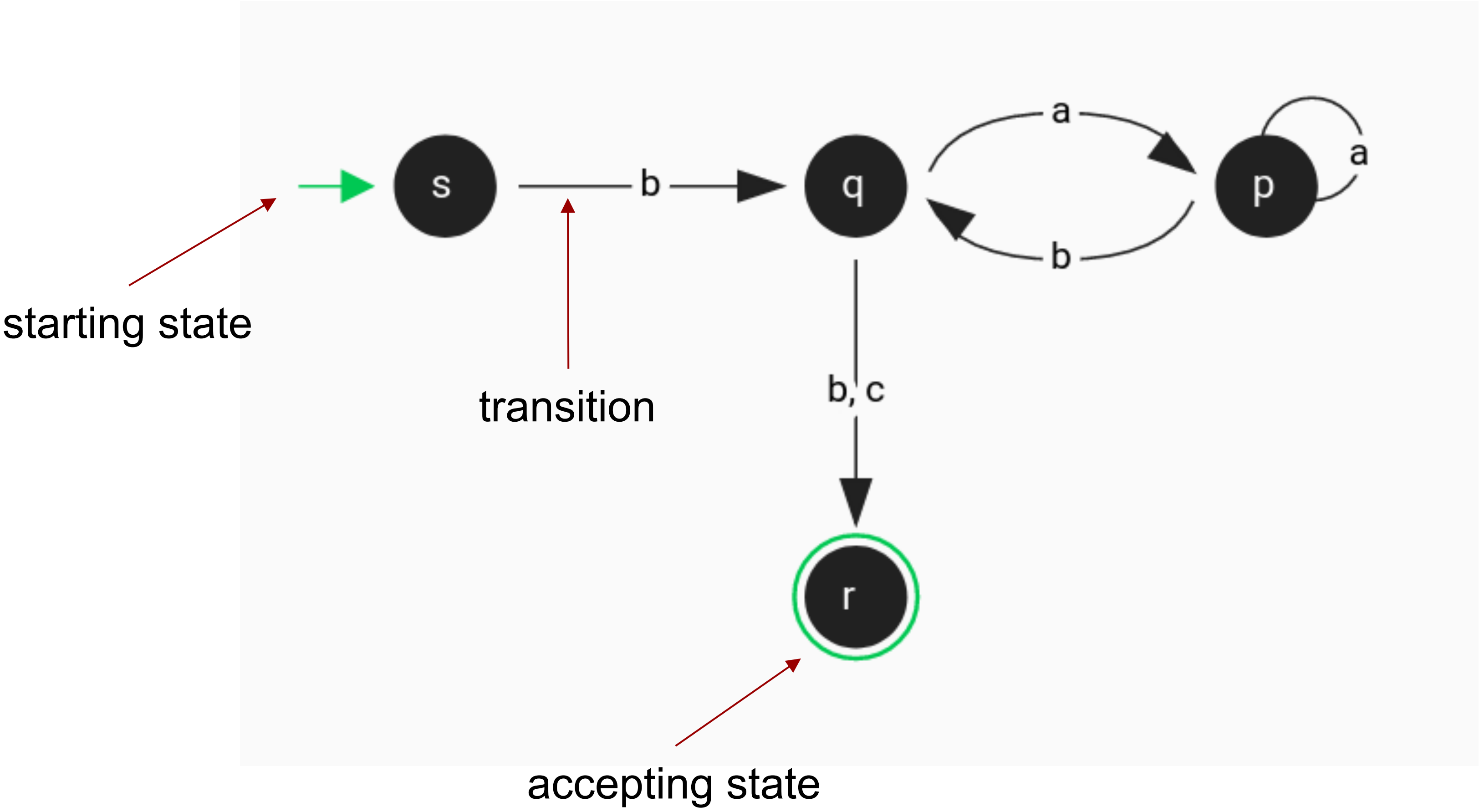


State Machine Models

- State Machines (automata) are abstract models that represent relevant features in their state and transitions
- States change at discrete points in time triggered by an input
- Example: simple light switch
    - two states: ‘on’ and ‘off’
    - input ‘flip’
- To apply this to security:
    - Characterize all states that fulfill security property
    - Check whether **all** state transitions preserve this property



Finite-State Machine



starting state

transition

accepting state

Alphabet: a, b, c

Why State Machine Models are useful for security analysis?

- Every possible state of a system is evaluated, showing all possible interactions between subjects and objects
- If every state is proven to be secure, the system is proven to be always secure!

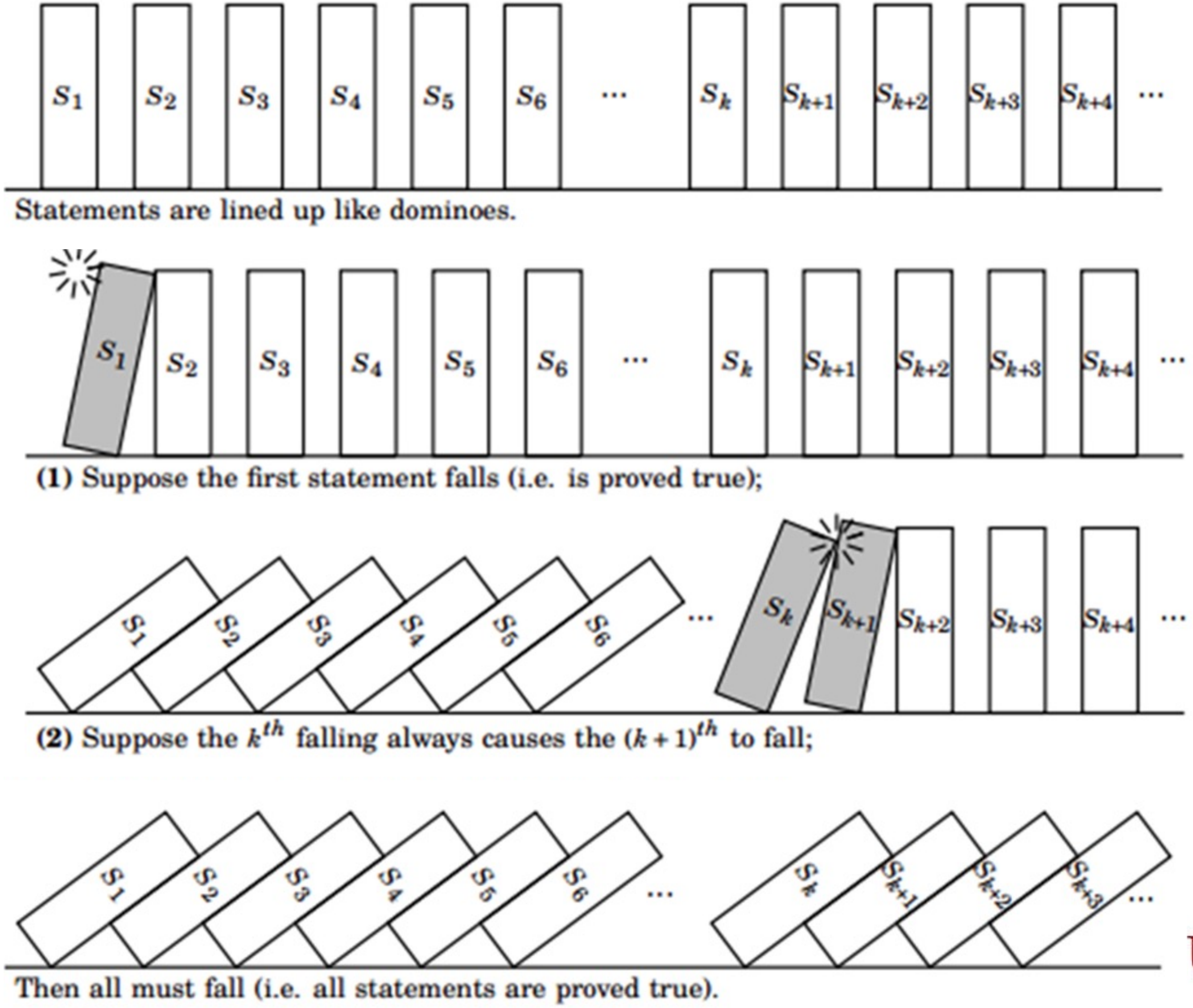
Outline

- Review
- Interpreting Reference Monitor components
- Motivation for formal security policy model
- Examples of policy models:
  - Access Control Matrix
  - Information flow models
  - State machine models, basic security theorem
- Bell-LaPadula (BLP) model
  - Preliminary concepts: inductive reasoning, MAC access classes, lattice structures

Is this security analysis practical?

- Every possible state of a system is evaluated, showing all possible interactions between subjects and objects
- If every state is proven to be secure, the system is proven to be always secure!
- Answer:
  - Proofs by Induction: is just like an ordinary proof in which every step must be justified
    - However it employs a neat trick which allows you to prove a statement about an arbitrary number!

Mathematical Induction is just like Domino Effect



Statements are lined up like dominoes.

(1) Suppose the first statement falls (i.e. is proved true);

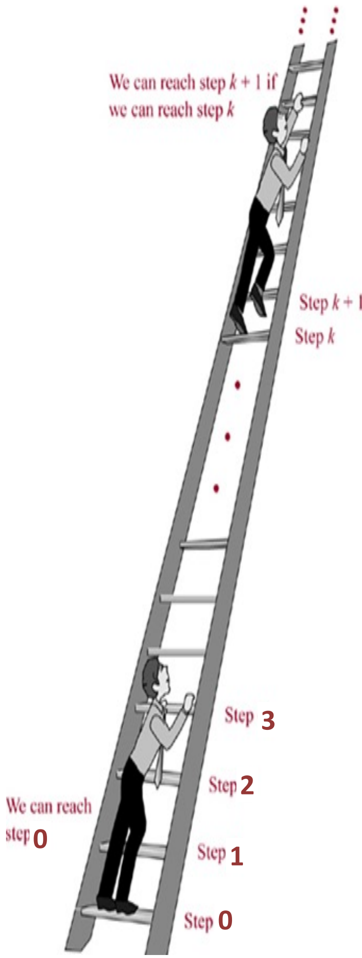
(2) Suppose the k^{th} falling always causes the $(k+1)^{th}$ to fall;

Then all must fall (i.e. all statements are proved true).

USC Viterbi
School of Engineering

Proof by Induction

- Three parts:
  - Base case: show it is true for one element
    - Can reach the first step
  - Show that it is true for the next element
    - From any one step you can reach the next step
  - Inductive hypothesis: assume it is true for any given element
    - Can reach the top of the ladder



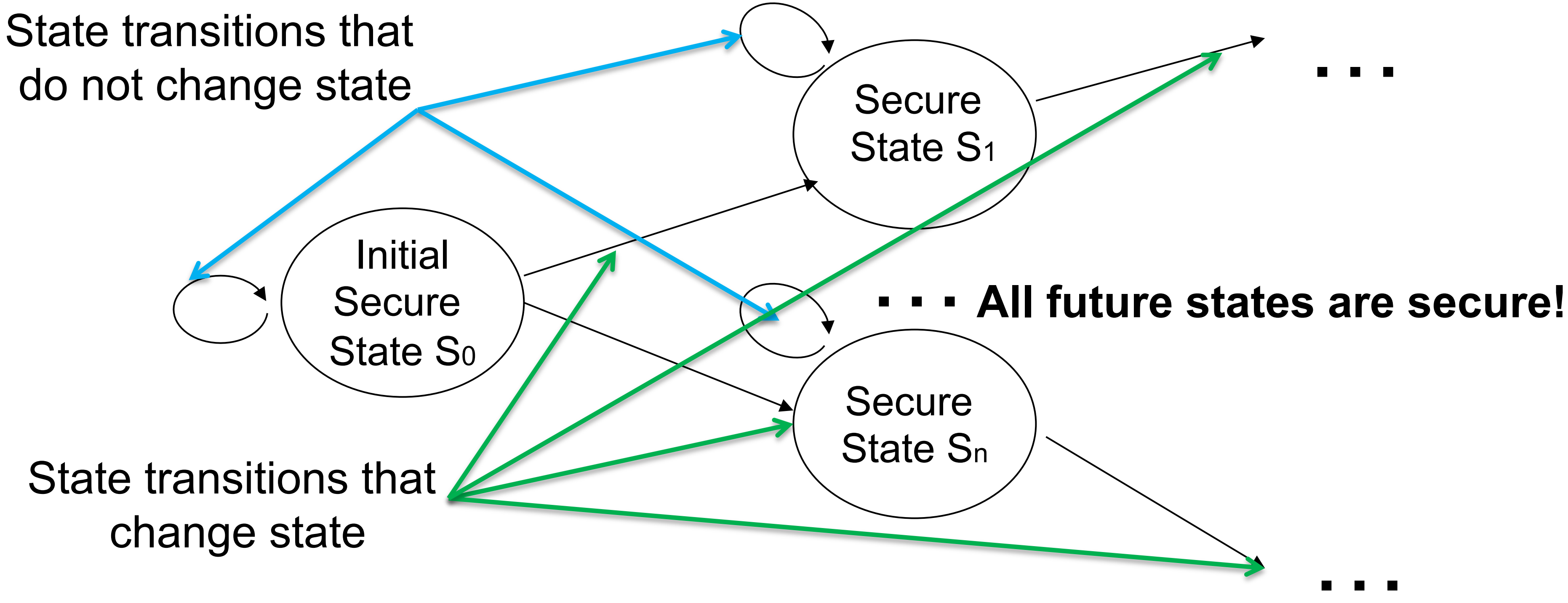
Let P(n) be the property that we can reach the n^{th} step
If P(0) is true and
for any $k \geq 1$, if P(k) is true then P(k + 1) is also true

$\longrightarrow$P(n) is true for all $n \geq 0$

Basic Security Theorem for State Machine Models

- To design a secure system:
  1. Define state set so that it captures ‘security’
  2. Check that **initial** state of the system is ‘secure’ state
  3. Check that all state transitions starting in a ‘secure’ state yield a ‘secure’ state
  4. Security is then preserved by all state transitions
     - The system will always be ‘secure’
- This Basic Security Theorem has been derived **without precise definition of “security”**
- This theorem is not related to particular security policy, it’s a feature of state machine models

Basic Security Theorem Pictorially


State transitions that do not change state
Secure State S1
Initial Secure State S0
. . .
. . . All future states are secure!
Secure State Sn
State transitions that change state
. . .

©Tatyana Ryutov

Case Study: Unix Setuid Model

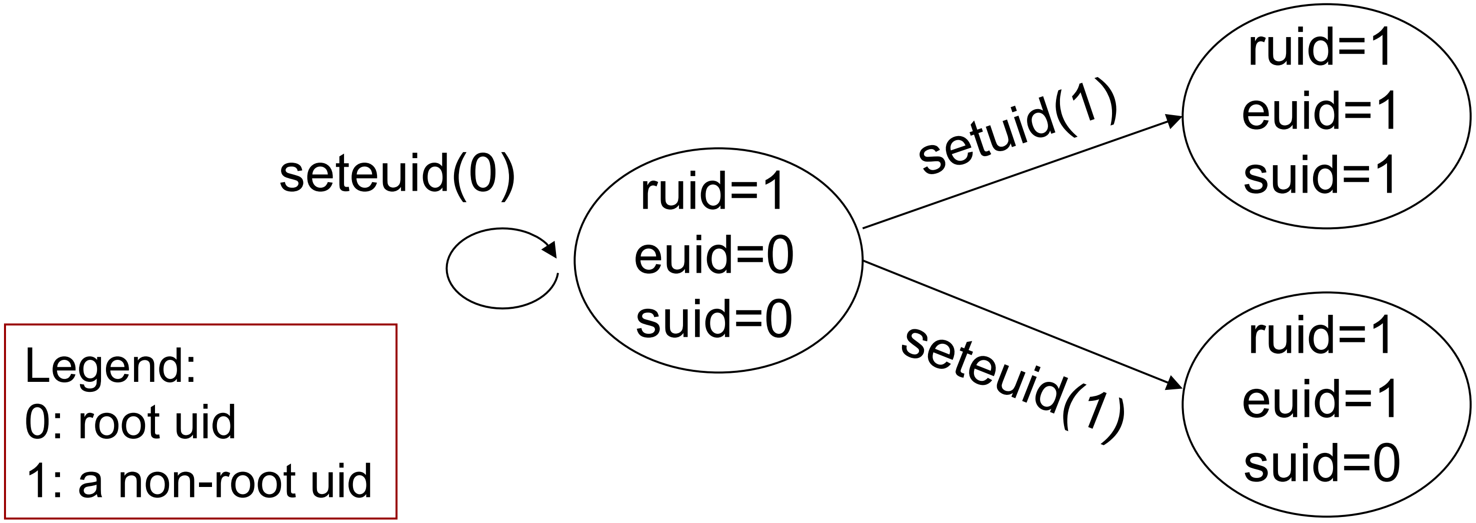
- Consider Unix Setuid formal model
  - H. Chen, D. Wagner and D. Dean, "Setuid Demystified," Proceedings of the11th USENIX Security Symposium, 2002, pp. 171-190
- Access control in Unix is based the UserID model
- Each process has 3 user IDs:
  - Real uid (ruid)
  - Effective uid (euid)
  - Saved uid (suid)
- Uid-setting system calls for a process to raise and drop privileges
  - setuid() seteuid() setreuid() setresuid()

The `setuid` Issues

- Uid-setting system calls are a semantic mess
  - Counter-intuitive semantics
  - Subtle differences among different calls
  - Incompatible semantics of the same call in different Unix systems (e.g., Linux, Solaris, FreeBSD)
  - Incomplete, inaccurate, or even wrong documentation
- Confusion has caused many security vulnerabilities
- Idea: use a formal model to describe the user ID model
  - Build an FSA (Final State Automata):
    - states describe user IDs of a process
    - transitions describe the semantics of uid-setting system calls

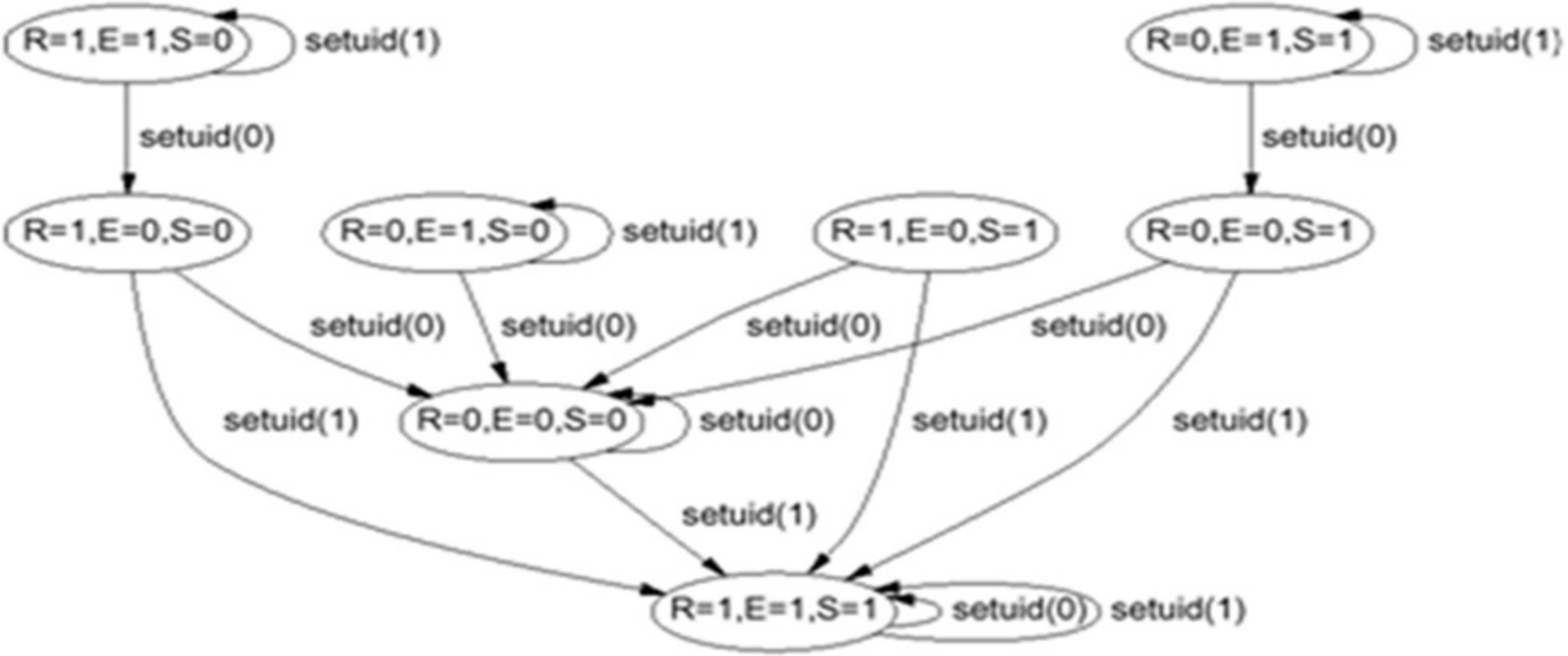
Formal Model of the Setuid API

- Finite State Automaton (FSA) model
  - States: describing the user IDs of a process
  - Transitions: describing the semantics of the setuid API calls



Legend:
0: root uid
1: a non-root uid

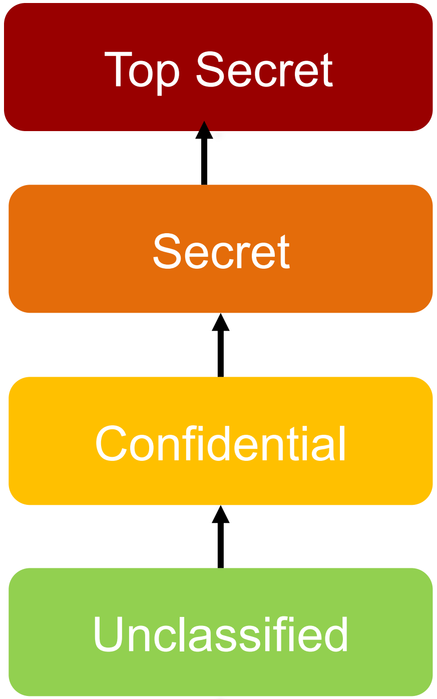
FSA for `setuid` in Linux



- FSA formally shows what `setuid` does

Recall: MAC Classification (Security Level)

- Top Secret (TS)
  - disclosure would cause “exceptionally grave damage” to national security
- Secret (S)
  - disclosure would cause “grave damage” to national security
- Confidential (C)
  - disclosure would cause “damage” or be “prejudicial” to national security
- Unclassified
  - not technically a classification level

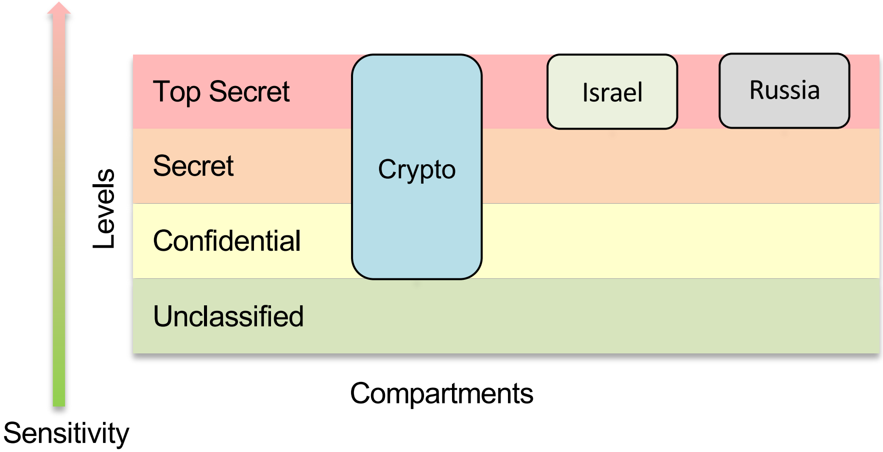


Notation: MAC Levels

- Security levels arranged in linear ordering
  - All levels can be compared
  - E.g., military policy for confidentiality
    - Top Secret > Secret > Confidential > Unclassified
  - E.g., corporate levels
    - Restricted > Management > Confidential > Public
- Subject levels consist of security clearance L(s)
- Objects have security classification L(o)

MAC Compartments (Categories)

- The **“need to know”** policy provides an orthogonal structure called compartmentalization
- Categories can be critical in complex coalitions
  - Example: the US may have two allies that do not wish to share information (e.g., Israel and Russia)
    - Policy must support:
      - Top Secret, Israel
      - Top Secret, Russia
      - Top Secret, Israel and Russia
- Security categories are independent and **non-comparable** (unordered)
  - E.g., the “Crypto” category for protecting keys
  - E.g., possible corporate categories: HR, Engineering
- Subjects and objects use the same category labels



Example: PitBull Sensitivity Levels and Categories

- PitBull is a low assurance multilevel security (MLS) operating system software product
  - Works in kernel at Linux Security Module (LSM)
- All PitBull label configurations stored under `/etc/security` directory and applied as extended attributes (XATTR) to the file systems and processes
  - System definition of security levels, categories, device labels, and user clearances
- Sensitivity Levels and Categories
  - 32,768 potential Sensitivity Level labels
  - 1,024 or 4,096 (RHEL 6.8 PitBull) potential Categories

More info: https://gdmissionsystems.com/products/multilevel-security/pitbull-trusted-operating-system

Notation: MAC Access Classes

- Clearance S and Classification O = “security level”
  - Also known as an “access class”
  - Each access class describes a kind of information
  - Access class can always be represented by a label
- Access class: **security level** (L) **categories** (C)
  - Single L value; C is a **finite set**
  - We write access class (L, C)

FSPM Abstraction for MAC Policy

- MAC reflects a set of rules for comparing labels
- Members of set of levels (L) form a total ordering
  - Two members can be compared, e.g.,<, ≤, =, etc.
- Members of category set (C) are non-comparable
- For access classes: (L, C) dominates (L′, C′)
  - **If and only if (IFF)** L′ ≤ L and C′ ⊆ C
  - Notation for access classes: (L, C) *dom* (L′, C′)

Example: MAC Policy Level & Category

- (L, C) *dom* (L', C') iff $L' \leq L$ and $C' \subseteq C$

- Examples:

  (Top Secret, {A, B}) ? (Secret, {A})

  (Secret, {A, B}) ? (Confidential, {A, B})

  (Top Secret, {A}) ? (Confidential, {B})

  (Top Secret, {∅}) ? (Confidential, {A, B})

Note: ¬*dom* does not mean that the opposite statement is true, e.g., (Top Secret, {A}) ¬*dom* (Confidential, {B}) does not mean (Confidential, {B}) *dom* (Top Secret, {A})

FSPM Abstraction for MAC Policy

- Dominate relation (*dom*) satisfies standard conditions:
  1. reflexivity
  2. anti-symmetry
  3. transitivity
- For a particular set of labels and relation *dom*
  1. x *dom* x
  2. x *dom* y Λ y *dom* x ➔ x = y, and
  3. x *dom* y Λ y *dom* z ➔ x *dom* z
- True for all access classes (and their labels)
- Labels that do not meet these are not suitable for MAC

MAC Access Classes (Security Labels)

- MAC reflects a set of rules for comparing labels
- Members of set of levels (L) form a total ordering
  - Two members can be compared, e.g.,<, ≤, =, etc.
- Members of category set (C) are non-comparable
- For access classes: (L, C) dominates (L′, C′)
  - If and only if (IFF) $L' \leq L$ and $C' \subseteq C$
  - Notation for access classes: **(L, C) *dom* (L′, C′)**

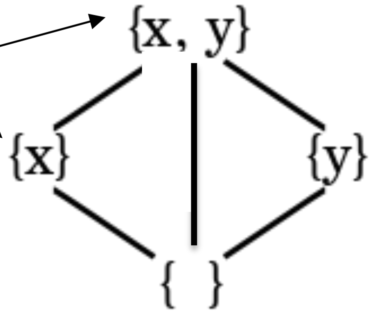
Lattice Structure

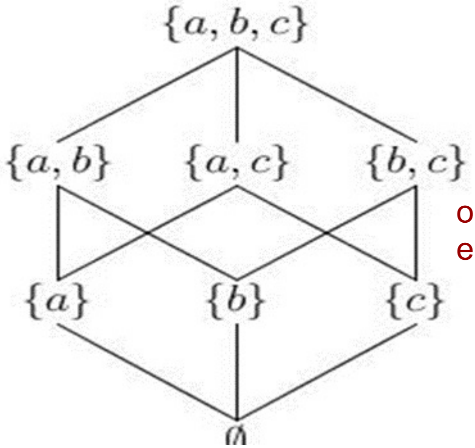
- **Total order** is linear (like a chain): a ≥ b, b ≥ c
- **Partial order** of a set occurs when a relation orders some, but not all elements of a set
  - Binary relation that is:
    1. Reflexive: a ≥ a
    2. Anti-symmetric: a ≥ b and b ≥ a then a = b
    3. Transitive: a ≥ b and b ≥ c then a ≥ c
- **Lattice**: a partially ordered finite set in which every two elements have a least upper bound (LUB) **and** a greatest lower bound (GLB)

Lattice Examples

- Lattice representation (Hasse diagram):
  - *dom* relationship is shown by undirected edges (assuming edges oriented downwards)
  - absence of a horizontal connection means inco

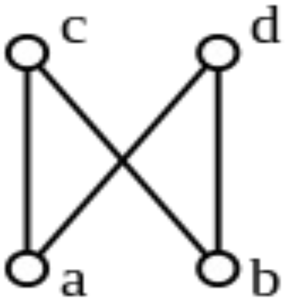
names of the nodes, not elements in a set

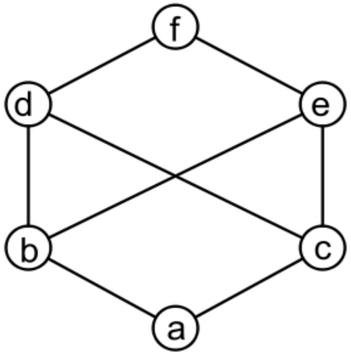




omit some vertical edges for clarity

- Most partial ordered sets are **not** lattices:



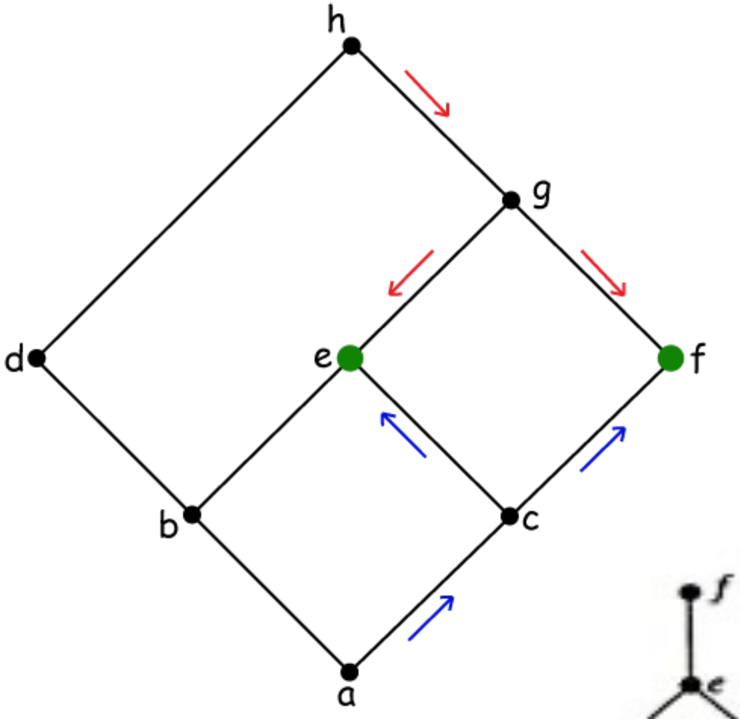


c and d: no LUB; a and b: no GLB

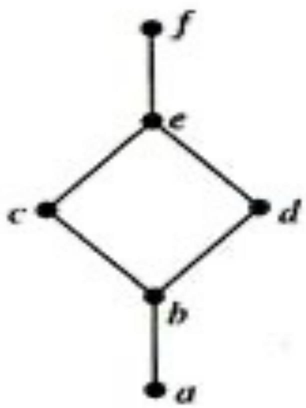
b and c: upper bounds d, e, and f, but none of them is LUB

USC Viterbi
School of Engineering

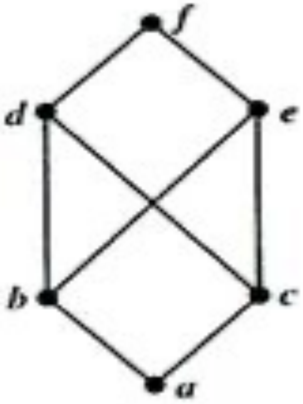
How can we determine GLB and LUB?



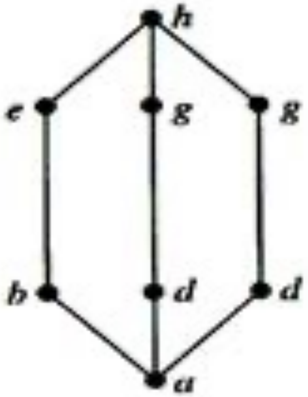
Which of these are lattices?



a.



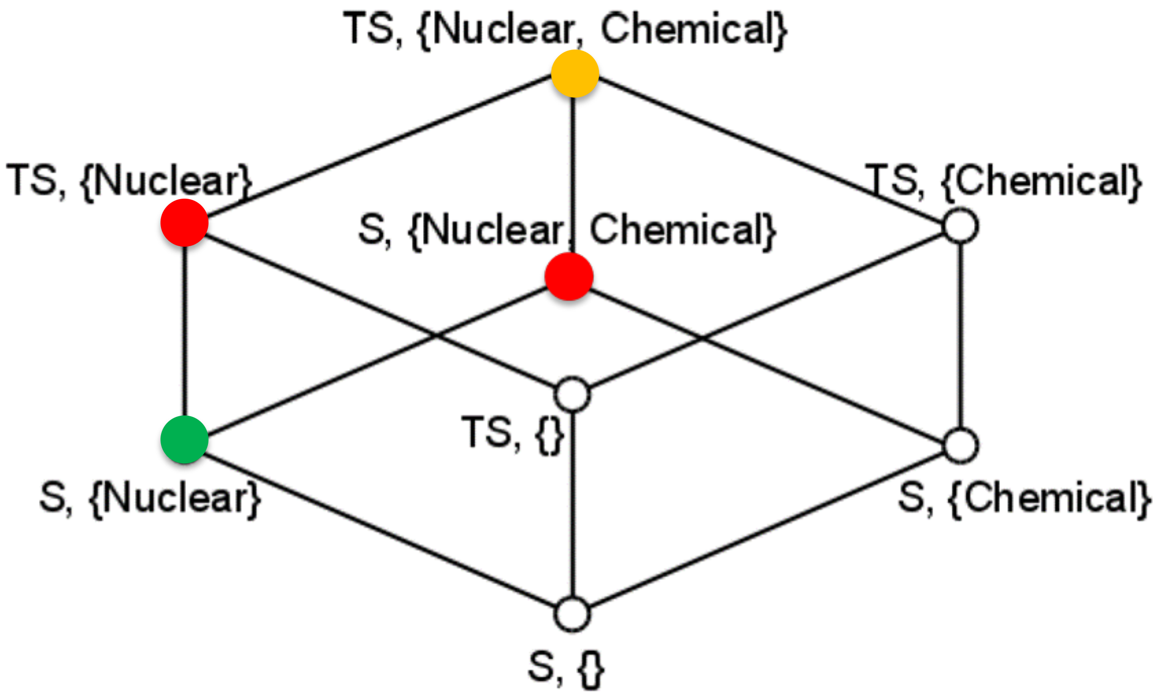
b



c

USC Viterbi
School of Engineering

BLP Lattice Example



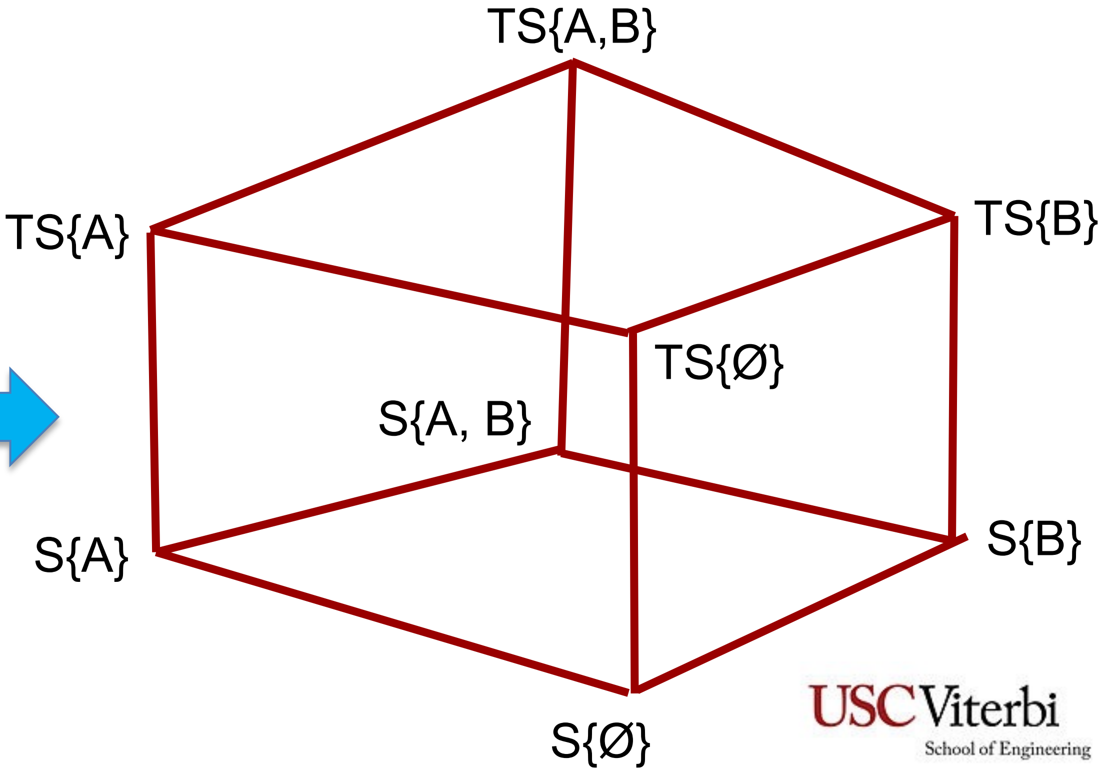
lub? $lub((TS, \{Nuclear\}), (S, \{Nuclear, Chemical\})) = (TS, \{Nuclear, Chemical\})$

glb? $glb((TS, \{Nuclear\}), (S, \{Nuclear, Chemical\})) = (S, \{Nuclear\})$

Example: Lattice Product

- Product of 2 lattices is a lattice
- Hierarchical classes with compartments (categories):
  - Levels: TS (top secret), S (secret), TS > S
  - Categories: A and B
  - Sets of categories: Ø, {A}, {B}, {A, B}

```
TS          {A,B}
|          /     \
S       {A}       {B}
           \     /
              Ø
```


TS{A,B}
TS{A}
TS{B}
TS{Ø}
S{A, B}
S{A}
S{B}
S{Ø}