Lecture 19, May 13, 2026

ECS 235B, Foundations of Computer and Information Security

Spring Quarter 2026

Review of Information Flow Notation

- Bell-LaPadula Model embodies information flow policy
  - Given compartments *A*, *B*, info can flow from *A* to *B* iff *B dom A*
- So does Biba Model
  - Given compartments *A*, *B*, info can flow from *A* to *B* iff *A dom B*
- Variables *x*, *y* assigned compartments $\underline{x}$, $\underline{y}$ as well as values
  - Confidentiality (Bell-LaPadula): if $\underline{x} = A$, $\underline{y} = B$, and *B dom A*, then *y* := *x* allowed but not *x* := *y*
  - Integrity (Biba): if $\underline{x} = A$, $\underline{y} = B$, and *A dom B*, then *x* := *y* allowed but not *y* := *x*
- What follows focuses on confidentiality (Bell-LaPadula)

Entropy and Information Flow

- Idea: information flows from *x* to *y* as a result of a sequence of commands *c* if you can deduce information about *x* before *c* from the value in *y* after *c*
- Formally:
  - *s* time before execution of *c*, *t* time after
  - $H(x_s \mid y_t) < H(x_s \mid y_s)$
  - If no *y* at time *s*, then $H(x_s \mid y_t) < H(x_s)$

Example 1

- Command is $x := y + z$; where:
  - x does not exist initially (that is, has no value)
  - $0 \le y \le 7$, equal probability
  - $z = 1$ with probability 1/2, $z = 2$ or 3 with probability 1/4 each
- s state before command executed; t, after; so
  - $H(y_s) = H(y_t) = -8(1/8) \lg (1/8) = 3$
- You can show that $H(y_s \mid x_t) = (3/32) \lg 3 + 9/8 \approx 1.274 < 3 = H(y_s)$
  - Thus, information flows from y to x

Example 2

- Command is

  if $x = 1$ **then** $y := 0$ **else** $y := 1$;

  where x, y equally likely to be either 0 or 1
- x can be either 0 or 1 with equal probability, so $H(x_s) = 1$
- If $y_t = 1$ then $x_s = 0$ (and vice versa), so $H(x_s \mid y_t) = 0$
  - Thus, $H(x_s \mid y_t) = 0 < 1 = H(x_s)$
- So information flowed from x to y

Implicit Flow of Information

- Information flows from *x* to *y* without an *explicit* assignment of the form *y* := *f*(*x*)
  - *f*(*x*) an arithmetic expression with variable *x*
- Example from previous slide:

if *x* = 1 **then** *y* := 0 **else** *y* := 1;

- So must look for implicit flows of information to analyze program

Notation

- $\underline{x}$ means class of x
  - In Bell-LaPadula based system, same as “label of security compartment to which x belongs”
- $\underline{x} \leq \underline{y}$ means “information can flow from an element in class of x to an element in class of y
  - Or, “information with a label placing it in class $\underline{x}$ can flow into class $\underline{y}$”

Information Flow Policies

Information flow policies are usually:

- reflexive
  - So information can flow freely among members of a single class
- transitive
  - So if information can flow from class 1 to class 2, and from class 2 to class 3, then information can flow from class 1 to class 3

Non-Transitive Policies

- Betty is a confident of Anne
- Cathy is a confident of Betty
  - With transitivity, information flows from Anne to Betty to Cathy
- Anne confides to Betty she is having an affair with Cathy's spouse
  - Transitivity undesirable in this case, probably

Non-Lattice Transitive Policies

- 2 faculty members co-PIs on a grant
  - Equal authority; neither can overrule the other
- Grad students report to faculty members
- Undergrads report to grad students
- Information flow relation is:
  - Reflexive and transitive
- But some elements (people) have no “least upper bound” element
  - What is it for the faculty members?

Confidentiality Policy Model

- Lattice model fails in previous 2 cases
- Generalize: policy $I = (SC_I, \leq_I, join_I)$:
  - SC_I set of security classes
  - $\leq_I$ ordering relation on elements of SC_I
  - $join_I$ function to combine two elements of SC_I
- Example: Bell-LaPadula Model
  - SC_I set of security compartments
  - $\leq_I$ ordering relation *dom*
  - $join_I$ function *lub*

Confinement Flow Model

- $(I, O, confine, \rightarrow)$
  - $I = (SC_I, \leq_I, join_I)$
  - O set of entities
  - $\rightarrow$: $O \times O$ with $(a, b) \in \rightarrow$ (written $a \rightarrow b$) iff information can flow from a to b
  - for $a \in O$, $confine(a) = (a_L, a_U) \in SC_I \times SC_I$ with $a_L \leq_I a_U$
    - Interpretation: for $a \in O$, if $x \leq_I a_U$, information can flow from x to a, and if $a_L \leq_I x$, information can flow from a to x
    - So a_L lowest classification of information allowed to flow out of a, and a_U highest classification of information allowed to flow into a

Assumptions, *etc.*

- Assumes: object can change security classes
  - So, variable can take on security class of its data
- Object x has security class $\underline{x}$ currently
- Note transitivity *not* required
- If information can flow from a to b, then b dominates a under ordering of policy I:

  $(\forall\ a, b \in O)[\ a \rightarrow b \Rightarrow a_L \leq_I b_U\ ]$

Example 1

- SC_I = { U, C, S, TS }, with U $\leq_I$ C, C $\leq_I$ S, and S $\leq_I$ TS
- $a, b, c \in O$
  - confine(a) = [C, C]
  - confine(b) = [S, S]
  - confine(c) = [TS, TS]
- Secure information flows: $a \rightarrow b$, $a \rightarrow c$, $b \rightarrow c$
  - As $a_L \leq_I b_U$, $a_L \leq_I c_U$, $b_L \leq_I c_U$
  - Transitivity holds

Example 2

- SC_I, $\leq_I$ as in Example 1
- $x, y, z \in O$
  - confine(x) = [C, C]
  - confine(y) = [S, S]
  - confine(z) = [C, TS]
- Secure information flows: $x \rightarrow y$, $x \rightarrow z$, $y \rightarrow z$, $z \rightarrow x$, $z \rightarrow y$
  - As $x_L \leq_I y_U$, $x_L \leq_I z_U$, $y_L \leq_I z_U$, $z_L \leq_I x_U$, $z_L \leq_I y_U$
  - Transitivity does not hold
    - $y \rightarrow z$ and $z \rightarrow x$, but $y \rightarrow x$ is false, because $y_L \leq_I x_U$ is false

Transitive Non-Lattice Policies

- Q = (S_Q, $\leq_Q$) is a *quasi-ordered set* when $\leq_Q$ is transitive and reflexive over S_Q
- How to handle information flow?
  - Define a partially ordered set containing quasi-ordered set
  - Add least upper bound, greatest lower bound to partially ordered set
  - It's a lattice, so apply lattice rules!

In Detail …

- $\forall x \in S_Q$: let $f(x) = \{ y \mid y \in S_Q \wedge y \leq_Q x \}$
  - Define $S_{QP} = \{ f(x) \mid x \in S_Q \}$
  - Define $\leq_{QP} = \{ (x, y) \mid x, y \in S_{QP} \wedge x \subseteq y \}$
    - S_{QP} partially ordered set under $\leq_{QP}$
    - f preserves order, so $y \leq_Q x$ iff $f(x) \leq_{QP} f(y)$
- Add upper, lower bounds
  - $S_{QP}' = S_{QP} \cup \{ S_Q, \varnothing \}$
  - Upper bound $ub(x, y) = \{ z \mid z \in S_{QP} \wedge x \subseteq z \wedge y \subseteq z \}$
  - Least upper bound $lub(x, y) = \cap ub(x, y)$
    - Lower bound, greatest lower bound defined analogously

And the Policy Is …

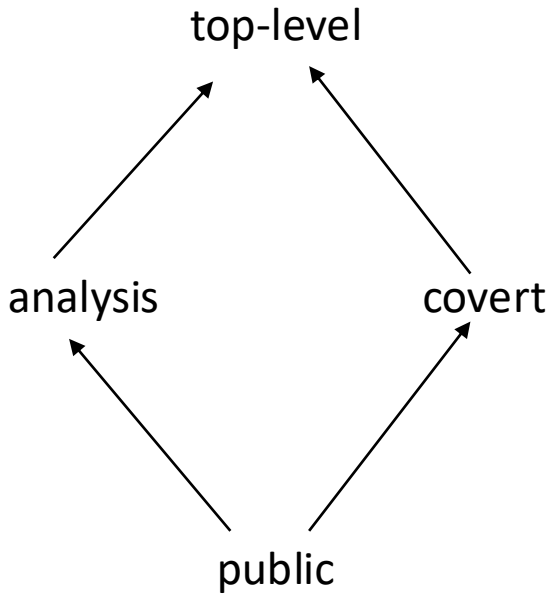
- Now (S_{QP}', $\leq_{QP}$) is lattice
- Information flow policy on quasi-ordered set emulates that of this lattice!

Nontransitive Flow Policies

- Government agency information flow policy (on next slide)
- Entities public relations officers PRO, analysts A, spymasters S
  - *confine*(PRO) = [public, analysis]
  - *confine*(A) = [analysis, top-level]
  - *confine*(S) = [covert, top-level]

Information Flow

- By confinement flow model:
  - PRO ≤ A, A ≤ PRO
  - PRO ≤ S
  - A ≤ S, S ≤ A
- Data *cannot* flow to public relations officers; not transitive
  - S ≤ A, A ≤ PRO
  - S ≤ PRO is *false*



Transforming Into Lattice

- Rough idea: apply a special mapping to generate a subset of the power set of the set of classes
  - Done so this set is partially ordered
  - Means it can be transformed into a lattice
- Can show this mapping preserves ordering relation
  - So it preserves non-orderings and non-transitivity of elements corresponding to those of original set

Dual Mapping

- $R = (SC_R, \leq_R, join_R)$ reflexive info flow policy
- $P = (S_P, \leq_P)$ ordered set
  - Define *dual mapping* functions $l_R, h_R: SC_R \rightarrow S_P$
    - $l_R(x) = \{ x \}$
    - $h_R(x) = \{ y \mid y \in SC_R \wedge y \leq_R x \}$
  - S_P contains subsets of SC_R; $\leq_P$ subset relation
  - Dual mapping function *order preserving* iff

$$(\forall a, b \in SC_R)[\ a \leq_R b \Leftrightarrow l_R(a) \leq_P h_R(b)\ ]$$

Theorem

Dual mapping from reflexive information flow policy R to ordered set P order-preserving

Proof sketch: all notation as before

($\Rightarrow$) Let $a \leq_R b$. Then $a \in l_R(a)$, $a \in h_R(b)$, so $l_R(a) \subseteq h_R(b)$, or $l_R(a) \leq_P h_R(b)$

($\Leftarrow$) Let $l_R(a) \leq_P h_R(b)$. Then $l_R(a) \subseteq h_R(b)$. But $l_R(a) = \{ a \}$, so $a \in h_R(b)$, giving $a \leq_R b$

Information Flow Requirements

- Interpretation: let *confine*(x) = [$\underline{x}_L$, $\underline{x}_U$], consider class $\underline{y}$
  - Information can flow from x to element of $\underline{y}$ iff $\underline{x}_L \leq_R \underline{y}$, or $l_R(\underline{x}_L) \subseteq h_R(\underline{y})$
  - Information can flow from element of $\underline{y}$ to x iff y $\leq_R \underline{x}_U$, or $l_R(\underline{y}) \subseteq h_R(\underline{x}_U)$

Revisit Government Example

- Information flow policy is *R*
- Flow relationships among classes are:

public $\leq_R$ public

public $\leq_R$ analysis

public $\leq_R$ covert

public $\leq_R$ top-level

analysis $\leq_R$ top-level

analysis $\leq_R$ analysis

covert $\leq_R$ covert

covert $\leq_R$ top-level

top-level $\leq_R$ top-level

Dual Mapping of *R*

- Elements l_R, h_R:
  l_R(public) = { public }
  h_R(public = { public }
  l_R(analysis) = { analysis }
  h_R(analysis) = { public, analysis }
  l_R(covert) = { covert }
  h_R(covert) = { public, covert }
  l_R(top-level) = { top-level }
  h_R(top-level) = { public, analysis, covert, top-level }

confine

- Let *p* be entity of type PRO, *a* of type A, *s* of type S
- In terms of *P* (not *R*), we get:
  - *confine*(*p*) = [{ public }, { public, analysis }]
  - *confine*(*a*) = [{ analysis }, { public, analysis, covert, top-level }]
  - *confine*(*s*) = [{ covert }, { public, analysis, covert, top-level }]

And the Flow Relations Are …

- $p \rightarrow a$ as $l_R(p) \subseteq h_R(a)$
  - $l_R(p)$ = { public }
  - $h_R(a)$ = { public, analysis, covert, top-level }
- Similarly: $a \rightarrow p$, $p \rightarrow s$, $a \rightarrow s$, $s \rightarrow a$
- But $s \rightarrow p$ is false as $l_R(s) \not\subset h_R(p)$
  - $l_R(s)$ = { covert }
  - $h_R(p)$ = { public, analysis }

Analysis

- $(S_P, \leq_P)$ is a lattice, so it can be analyzed like a lattice policy
- Dual mapping preserves ordering, hence non-ordering and non-transitivity, of original policy
  - So results of analysis of $(S_P, \leq_P)$ can be mapped back into $(SC_R, \leq_R, join_R)$

Confinement: Example Problem

- Server balances bank accounts for clients
- Server security issues:
  - Record correctly who used it
  - Send *only* balancing info to client
- Client security issues:
  - Log use correctly
  - Do not save or retransmit data client sends

Generalization

- Client sends request, data to server
- Server performs some function on data
- Server returns result to client
- Access controls:
  - Server must ensure the resources it accesses on behalf of client include *only* resources client is authorized to access
  - Server must ensure it does not reveal client's data to any entity not authorized to see the client's data

Confinement Problem

- Problem of preventing a server from leaking information that the user of the service considers confidential

Total Isolation

- Process cannot communicate with any other process
- Process cannot be observed

Impossible for this process to leak information

- Not practical as process uses observable resources such as CPU, secondary storage, networks, etc.

Example

- Processes *p*, *q* not allowed to communicate
  - But they share a file system
- Communications protocol:
  - *p* sends a bit by creating a file called *0* or *1*, then a second file called *send*
    - *p* waits until *send* is deleted before repeating to send another bit
  - *q* waits until file *send* exists, then looks for file *0* or *1*; whichever exists is the bit
    - *q* then deletes *0*, *1*, and *send* and waits until *send* is recreated before repeating to read another bit

Covert Channel

- A path of communication not designed to be used for communication
- In example, file system is a (storage) covert channel

Rule of Transitive Confinement

- If *p* is confined to prevent leaking, and it invokes *q*, then *q* must be similarly confined to prevent leaking
- Rule: if a confined process invokes a second process, the second process must be as confined as the first

Lipner's Notes

- All processes can obtain rough idea of time
  - Read system clock or wall clock time
  - Determine number of instructions executed
- All processes can manipulate time
  - Wait some interval of wall clock time
  - Execute a set number of instructions, then block

Approach: Isolation

- Constrain process execution in such a way it can only interact with other entities in a manner preserving isolation
  - Hardware isolation
  - Virtual machines
  - Library operating systems
  - Sandboxes
- Modify program or process so that its actions will preserve isolation
  - Program rewriting
  - Compiling
  - Loading

Hardware Isolation

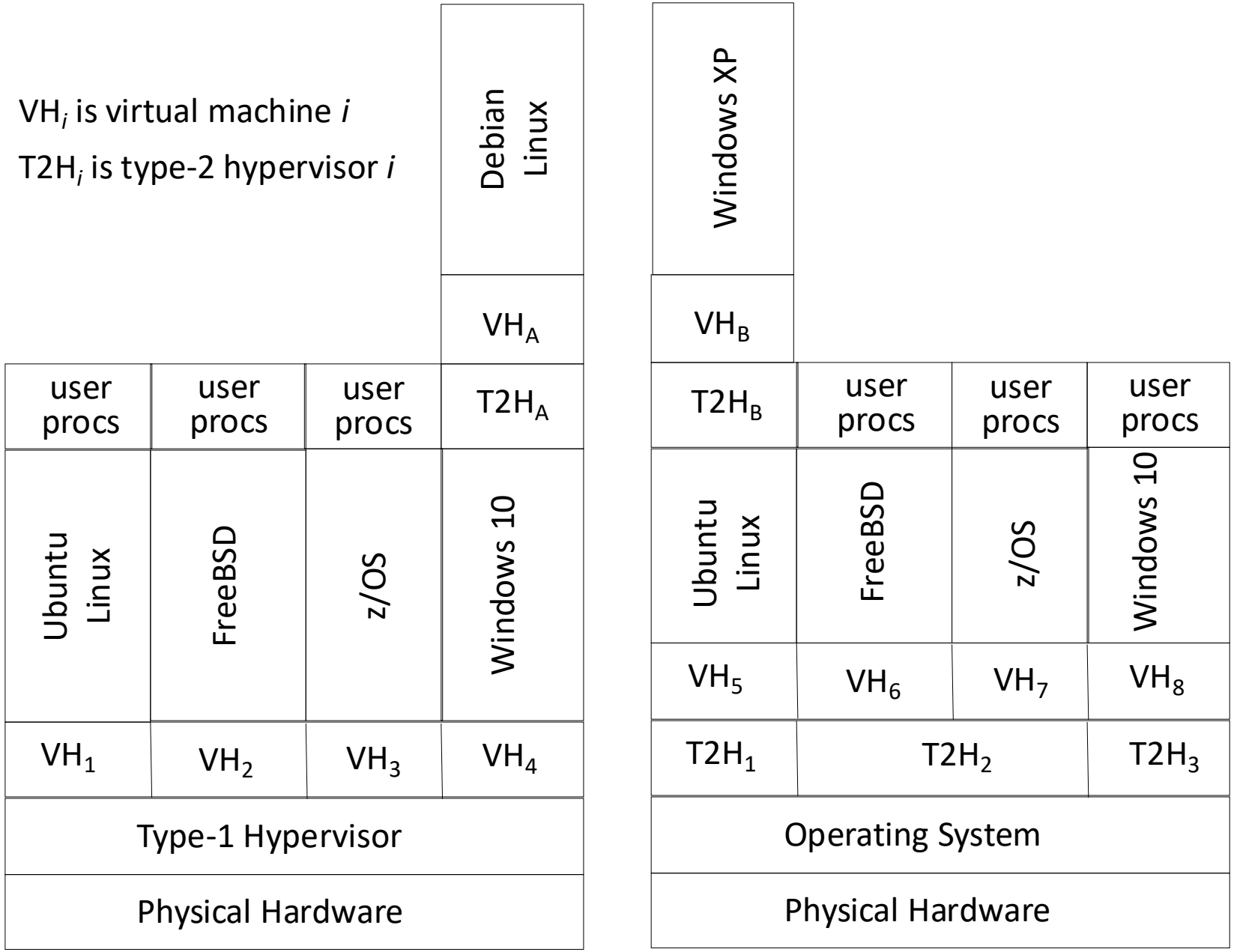
- Ensure the hardware is disconnected from any other system
  - This includes networking, including wireless
- Example: SCADA systems
  - 1st generation: serial protocols, not connected to other systems or networks; no security defenses needed, focus being on malfunctions
  - 2nd generation: serial networks connected to computers not connected to Internet
  - 3rd generation: TCP/IP protocol running on networks connected to Internet; need security defenses for attackers coming in over Internet
- Example: electronic voting systems
  - Physical isolation protects systems from attackers changing votes remotely
  - Required in many U.S. states, such as California: never connect them to any network

Virtual Machine

- Program that simulates hardware of a machine
  - Machine may be an existing, physical one or an abstract one
  - Uses special operating system, called *virtual machine monitor* (*VMM*) or *hypervisor*, to provide environment simulating target machine
- Types of virtual machines
  - Type 1 hypervisor: runs directly on hardware
  - Type 2 hypervisor: runs on another operating system
- Existing OSes do not need to be modified
  - Run under VMM, which enforces security policy
  - Effectively, VMM is a security kernel

What Is It?

- *Virtual machine monitor* (VMM) or *hypervisor* virtualizes system resources
  - Provides interface to give each program running on it the illusion that it is the only process on the system and is running directly on hardware
  - Provides illusion of contiguous memory beginning at address 0, a CPU, and secondary storage to *each* program
- *Type-1 hypervisor* runs directly on the hardware
- *Type-2 hypervisor* runs as a process on a regular operating system


VH$_i$ is virtual machine *i*
T2H$_i$ is type-2 hypervisor *i*
Debian Linux
Windows XP
VH$_A$
VH$_B$
user procs
user procs
user procs
T2H$_A$
T2H$_B$
user procs
user procs
user procs
Ubuntu Linux
FreeBSD
z/OS
Windows 10
Ubuntu Linux
FreeBSD
z/OS
Windows 10
VH$_1$
VH$_2$
VH$_3$
VH$_4$
VH$_5$
VH$_6$
VH$_7$
VH$_8$
T2H$_1$
T2H$_2$
T2H$_3$
Type-1 Hypervisor
Operating System
Physical Hardware
Physical Hardware

Privileged Instructions

1. VMM runs VM with operating system *o*, which is running process *p*
   - *p* tries to read, so privileged operation traps to hardware
2. VMM invoked, determines trap occurred in VM
   - VMM updates state of VM to make it look like hardware invoked *o* directly, so *o* tries to read, causing trap
3. VMM does read
   - Updates VM to make it seem like *o* did read
   - Transfers control to *o*

Privileged Instructions

4. *o* tries to switch context to *p*, causing another trap
5. VMM updates VM running *o* to make it appear *o* did context switch successfully
   - Transfers control to *o*, which (as *o* apparently did a context switch to *p*) returns control to *p*

Privileged Instructions