CS 5430

Information-Flow Policies

Prof. Clarkson

Spring 2016

Review: Access control

- Discretionary access control (DAC)
  - **Philosophy:** users have the *discretion* to specify policy themselves
  - Commonly, information belongs to the **owner** of object
  - Access control lists, privilege lists, capabilities
- Mandatory access control (MAC)
  - **Philosophy:** central authority *mandates* policy
  - Information belongs to the authority, not to the individual users
  - MLS and BLP, Chinese wall, Clark-Wilson, etc.

Limits of access control

Access control policies regulate **release** but not **propagation** of information

- Compare "can't read file" to "can't learn information in that file"
- MLS attempts to regulate propagation with "no write down"
  - But **trusted subjects** needed to declassify
  - Those could be buggy or malicious...

Limits of access control

[Lampson 1973] Malicious program could:

- Have **backdoor interface**: save secret information, later leak it to attacker who knows how to invoke interface
- Write information into a **file owned by attacker**
- Write information into a **public temp file**
- Use **IPC** to communicate with process run by attacker
- Leak information in **metadata** (billing reports, nonces chosen in protocols, ...)
- Use **shared resources** and OS API to encode information (e.g., file locking, CPU cycles)

Limits of access control

- Channel: means to communicate information
- Legitimate channel: intended for communication between programs
- Storage channel: written by one program and read by another
- Covert channel: not intended for information transfer yet exploitable for that purpose

Want information-flow control **along** channels, not just access control **to** channel

Information-flow control

[Denning 1976]

Secure information flow: no unauthorized flow of information is possible

- **Example:** BLP model of MLS
  - Information flow in that it prohibits flow of information with "no read up" and "no write down"
  - Access control in that it regulates access to objects at certain levels but not flow of information in trusted subjects
- **More examples:** this lecture and next

Information-flow control

Model:

- Set S of **subjects**
- Set O of **objects**
- Set L of security **labels**
- Function L(X) that gives label of entity (subject or object) X
  - labels might be static: don't change throughout execution
  - or dynamic: label of entity changes based on history of execution

Information-flow control

Model (continued):

- Function + that **combines** security labels:
  - $\ell_1 + \ell_2$ is label of information derived from ℓ_1 and ℓ_2
  - + is associative and commutative
- Relation $\rightarrow$ that **specifies what flows are allowed**:
  - If $\ell_1 \rightarrow \ell_2$ then information with label ℓ_1 is allowed to flow to ℓ_2
  - Along any legitimate or storage channels; ignore covert channels

Information-flow control

Common requirements on L and $\rightarrow$

- $\rightarrow$ is reflexive, transitive, antisymmetric
  - antisymmetry: if $\ell_1 \rightarrow \ell_2$ and $\ell_2 \rightarrow \ell_1$ then $\ell_1 = \ell_2$
- for all ℓ_1 and ℓ_2, there exists a least upper bound ℓ_3:
  - ℓ_3 is an upper bound: $\ell_1 \rightarrow \ell_3$ and $\ell_2 \rightarrow \ell_3$
  - ℓ_3 is the least upper bound: there is no ℓ such that
    - ℓ is an upper bound of ℓ_1 and ℓ_2
    - and $\ell \rightarrow \ell_3$
  - **define** $\ell_1 + \ell_2$ to be the least upper bound of ℓ_1 and ℓ_2

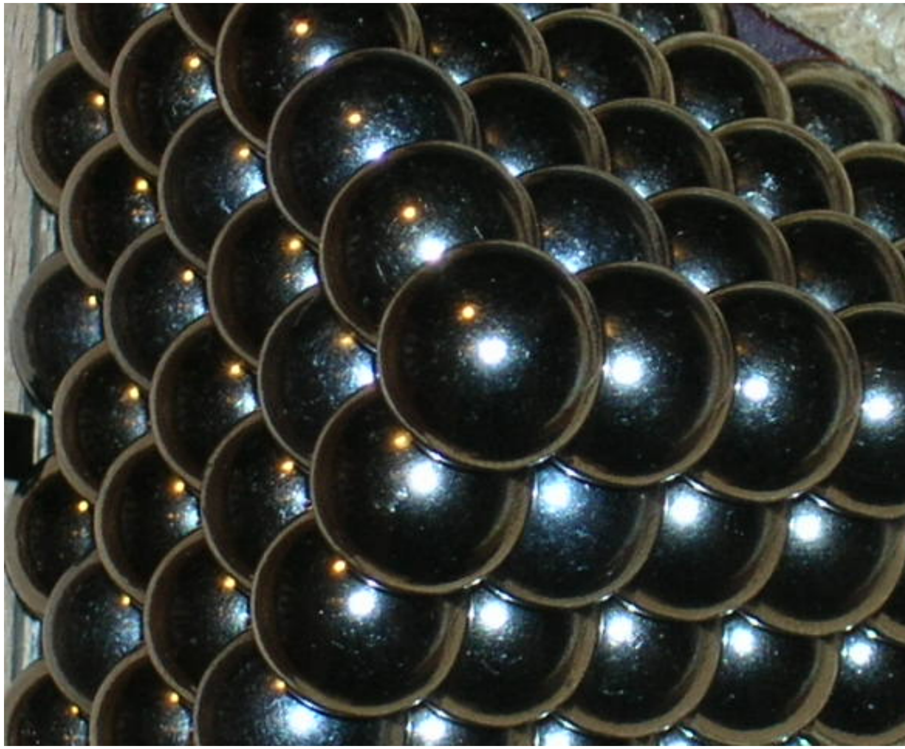
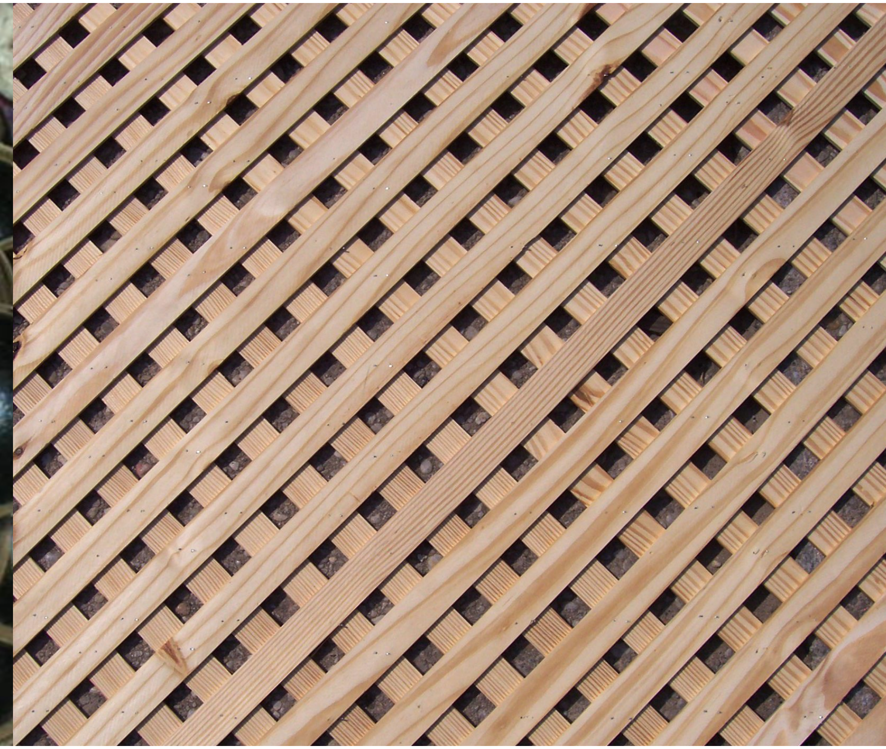
Information-flow control

Common requirements on L and $\rightarrow$ (continued)

- *for all ℓ_1 and ℓ_2, there exists a greatest lower bound ℓ_0:*
  - *ℓ_0 is a lower bound: $\ell_0 \rightarrow \ell_1$ and $\ell_0 \rightarrow \ell_2$*
  - *ℓ_0 is the greatest lower bound: there is no ℓ such that*
    - *ℓ is a lower bound of ℓ_1 and ℓ_2*
    - *and $\ell_0 \rightarrow \ell$*
  - *define $\ell_1 \times \ell_2$ to be the greatest lower bound of ℓ_1 and ℓ_2*
- there is a bottom label $\perp$: for all ℓ, $\perp \rightarrow \ell$
- there is a top label $\top$: for all ℓ, $\ell \rightarrow \top$

i.e., (L, $\rightarrow$) is a bounded lattice

Information-flow control



i.e., (L, $\longrightarrow$) is a bounded lattice

Example lattice: MLS

- Sens = {Unclassified, Confidential, Secret, Top Secret)
- Comp = set of all compartments, e.g., {nuclear,crypto}
- Comps = power set of all compartments, e.g., { {}, {nuclear}, {crypto}, {nuclear,crypto} }
- L = Sens x Comps
- (s1,c1) + (s2,c2) = (max(s1,s2), c1 $\cup$ c2)
- $\perp$ = (Unclassified, {})
- $\top$ = (Top Secret, Comp)
- $\rightarrow$ is the $\sqsubseteq$ ordering from MLS

Example lattice: two point

- $L = \{low, high\}$
- $\ell_1 + \ell_2 =$
  - low if $\ell_1 = \ell_2 = low$
  - high o.w.
- $\perp$ = low
- $\top$ = high
- low $\rightarrow$ high, low $\rightarrow$ low, high $\rightarrow$ high

- think of this as MLS with only...
  - Unclassified (low) and Top Secret (high)
  - no compartments
- simple and captures important ideas, so use of two-point lattice is standard in information-flow literature

Information-flow control

A system has **secure information flow** iff its execution never causes an information flow that violates $\rightarrow$

- Given objects $a_1, \ldots, a_n$
- Compute new value $v = f(a_1, \ldots, a_n)$ with function f
- And v flows to object b
- If b's label is **static**, then $L(a_1)+\ldots+L(a_n) \rightarrow L(b)$ must hold
- If b's label is **dynamic**, then L(b) must be updated such that $L(a_1)+\ldots+L(a_n) \rightarrow L(b)$ holds

Information-flow control

A system has **secure information flow** iff its execution never causes an information flow that violates →

- Given objec
- Compute new … a_n) with function f
- And v flows to
- If b's label is **static**, then $L(a_1)+\ldots+L(a_n) \rightarrow L(b)$ must hold
- If b's label is **dynamic**, then L(b) must be updated such that $L(a_1)+\ldots+L(a_n) \rightarrow L(b)$ holds

How to determine?

Security conditions

- aka *security policies* or *security properties*
- Conditions that must hold of system execution to ensure secure information flow
- Grandfather of all information-flow security conditions: **noninterference**

Noninterference

[Goguen and Meseguer 1982]: Commands of high security users have no effect on observations of low security users

Let's make that precise…

Execution model

[Mantel 2003]

- An event is an atomic action of system:
  - Input (receive a message)
  - Output (send a message)
  - Internal (computation step)
- A trace is a sequence of execution events
  - <> is the empty trace
  - <e1 . e2> is the trace containing event e1 followed by e2
  - t1 . t2 is trace t1 followed by trace t2

Execution model

Model of system: set of possible traces

- Semantic model of system: based on behavior
- Not syntactic: based on program text
- Assume attacker knows entire set of traces
  - i.e., knows possible behaviors of system
  - worst case assumption

Execution model

Example: random number generator

- Suppose it can generate any length sequence of any integer
- Let Rand be its set of traces
  - <> in Rand
  - if t in Rand then for all integers n, <out(n)>.t in Rand

Execution model

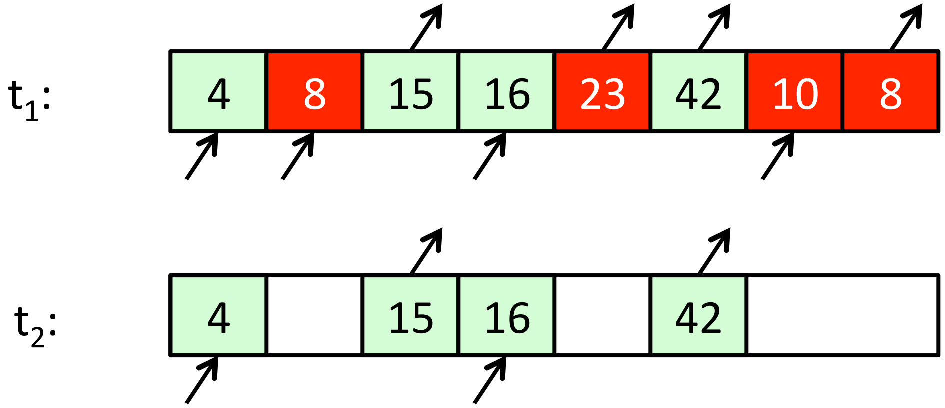
Example: leaky system

- After number received as high input, immediately produces that number as low output
- Let Leak be its set of traces
  - <> in Leak
  - if t in Leak then for all integers n, <hi-in(n) . low-out(n)>.t in Leak

Noninference

Intuition:

- anything that could happen in the presence of high events could also happen without them
- so nothing can be inferred about their occurrence



Noninference

[O'Halloran 1990, McLean 1994, Zakinthinos and Lee 1997, Mantel 2003]

- Assume sets Low and High of low and high events
  - i.e., two-point lattice
- Define proj(t, Low) as the projection of t to low events
  - i.e., delete all events that aren't low
  - proj(<>, Low) = <>
  - if e in Low then proj(<e>.t, Low) = <e>.proj(t, Low)
  - if e not in Low then proj(<e>.t, Low) = proj(t, Low)
- A trace set T satisfies **noninference** if
  for all t ∈ T, proj(t, Low) ∈ T

Noninference

Noninference **does not hold** of Leak

- (which is a good thing!)
- let Low = { <low-out(n)> | n }
- <hi-in(42) . low-out(42)> in Leak
- But <low-out(42)> not in Leak

Noninference

Generalized to an arbitrary lattice...

- A trace set T satisfies noninference if for all t in T and for all labels ℓ, proj-below(t, ℓ) ∈ T
- where proj-below(t, ℓ) is the projection of t to events at level ℓ or lower in lattice
  - i.e., any label ℓ' such that ℓ' → ℓ

Noninference

Noninference **does not protect** against leakage of *nonoccurrence* of events

- (which might not be a good thing)
- let T = { <hi-in(1), low-out(1)>, <hi-in(0), low-out(0)>,
  <low-out(1)>, <low-out(0)> }
- T satisfies noninference
- But low observer who sees low-out(1) can infer that hi-in(0) did not occur

Separability

Intuition: system behaves as though low and high parts are physically separated into two pieces (e.g., simulated *airgap*)

Separability

[McLean 1994, Zakinthinos and Lee 1997, Mantel 2003]

- Define proj(t, High) as the projection of t to high events
- Define interleaving(t1, t2) as the set of all traces that results from mixing events of t1 and t2 together
  - interleaving(t1, <>) = {t1}
  - interleaving(<>, t2) = {t2}
  - interleaving(<e1>.t1, <e2>.t2) =
    { <e1> . t' | t' ∈ interleaving(t1, <e2>.t2) }
    ∪
    { <e2> . t' | t' ∈ interleaving(<e1>.t1, t2) }

Separability

[McLean 1994, Zakinthinos and Lee 1997, Mantel 2003]

A trace set T satisfies separability if
for all tl, th ∈ T,
interleaving(proj(tl, Low), proj(th, High)) ⊆ T

Separability

Separability **does not hold** of Leak

- (which is a good thing!)
- <hi-in(42) . low-out(42)> and <hi-in(41) . low-out(41)> both in Leak
- But <hi-in(42) . low-out(41)> not in Leak

Separability

Generalized to an arbitrary lattice...

- A trace set T satisfies separability if for all tl, th $\in$ T, and for all security levels ℓ, interleaving(proj-below(tl, ℓ), proj-notbelow(th, ℓ)) $\subseteq$ T
- where proj-notbelow(t, ℓ) is the projection of t to events not below ℓ
  - i.e., any label ℓ' such that not($\ell' \rightarrow \ell$)
  - "proj-above" would be intuitive if a little inaccurate

Separability

Separability **prohibits useful flow** from low to high

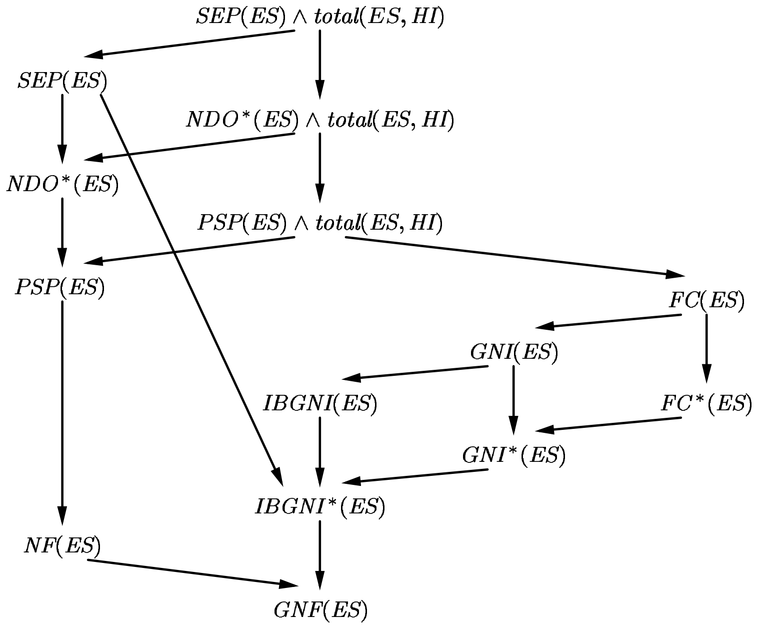
- (which is not a good thing)
- Consider system Log that logs all low inputs to high output
- <low-in(42) . hi-out(42)> and <low-in(41) . hi-out(41)> both in Log
- But <low-in(42) . hi-out(41)> not in Log
- Even though low observer doesn't learn anything about high inputs

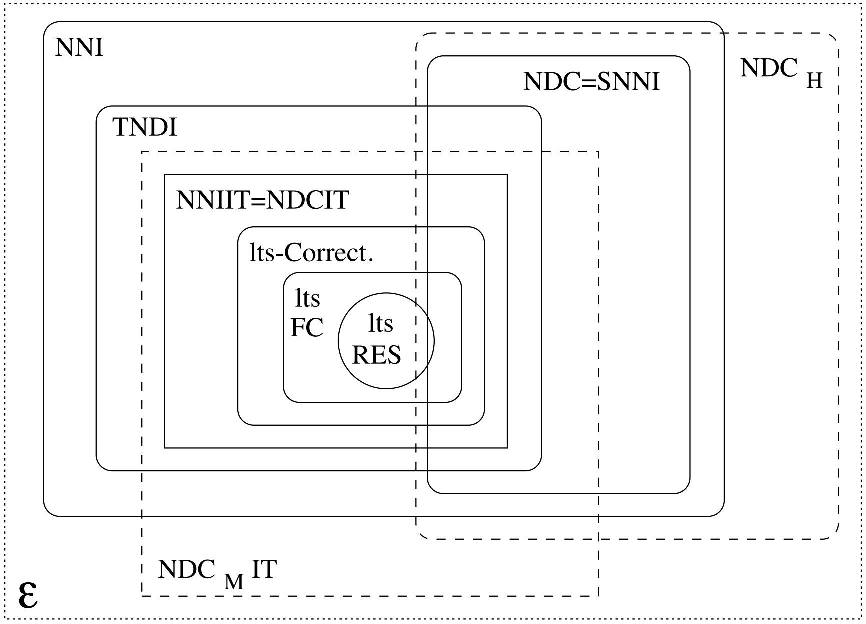
Noninference vs. separability

- Separability implies noninference [Mantel 2003]
- Neither is "perfect"
  - Prefer to rule out observability of nonoccurrence?
  - Or flows from low to high?

Variants of noninterference

Many others invented and compared
[Mantel 2003, Focardi and Gorrieri 2000]:





Variants of noninterference

I'm guilty too.

- [O'Neill, Clarkson, Chong 2006]: a variant of probabilistic noninterference
- [Micinski, Fetter-Degges, Jeon, Foster, Clarkson 2015]: noninterference for Android apps

Other challenges for noninterference

- **Compositionality:** two noninterfering systems when hooked up together might be interfering
- **Refinement:** a noninterfering specification might be refined to an interfering program
- **Declassification:** policies so far don't permit it; one solution is to make flow relation $\rightarrow$ be intransitive
- **Covert channels:** anything not part of the model is effectively a covert channel

Upcoming events

- [Wed] A6 out?

Don't let school interfere with your education.
– Mark Twain