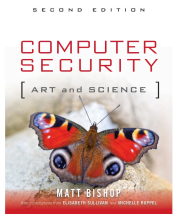
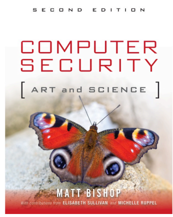


Noninterference and Policy Composition

Chapter 9

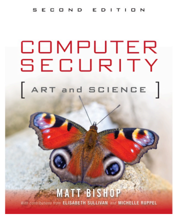
Overview

- Problem
  - Policy composition
- Noninterference
  - HIGH inputs affect LOW outputs
- Nondeducibility
  - HIGH inputs can be determined from LOW outputs
- Restrictiveness
  - When can policies be composed successfully



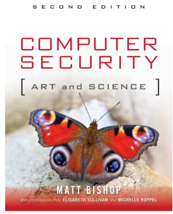
Composition of Policies

- Two organizations have two security policies
- They merge
  - How do they combine security policies to create one security policy?
  - Can they create a coherent, consistent security policy?



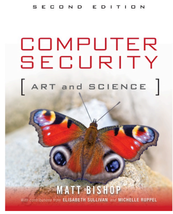
The Problem

- Single system with 2 users
  - Each has own virtual machine
  - Holly at system high, Lara at system low so they cannot communicate directly
- CPU shared between VMs based on load
  - Forms a *covert channel* through which Holly, Lara can communicate



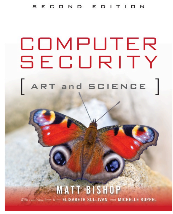
Example Protocol

- Holly, Lara agree:
  - Begin at noon
  - Lara will sample CPU utilization every minute
  - To send 1 bit, Holly runs program
    - Raises CPU utilization to over 60%
  - To send 0 bit, Holly does not run program
    - CPU utilization will be under 40%
- Not “writing” in traditional sense
  - But information flows from Holly to Lara



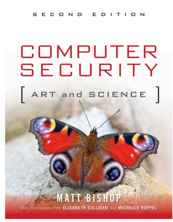
Policy vs. Mechanism

- Can be hard to separate these
- In the abstract: CPU forms channel along which information can be transmitted
  - Violates *-property
  - Not "writing" in traditional sense
- Conclusion:
  - Bell-LaPadula model does not give sufficient conditions to prevent communication, *or*
  - System is improperly abstracted; need a better definition of "writing"



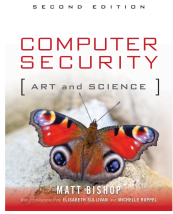
Composition of Bell-LaPadula

- Why?
  - Some standards require secure components to be connected to form secure (distributed, networked) system
- Question
  - Under what conditions is this secure?
- Assumptions
  - Implementation of systems precise with respect to each system’s security policy

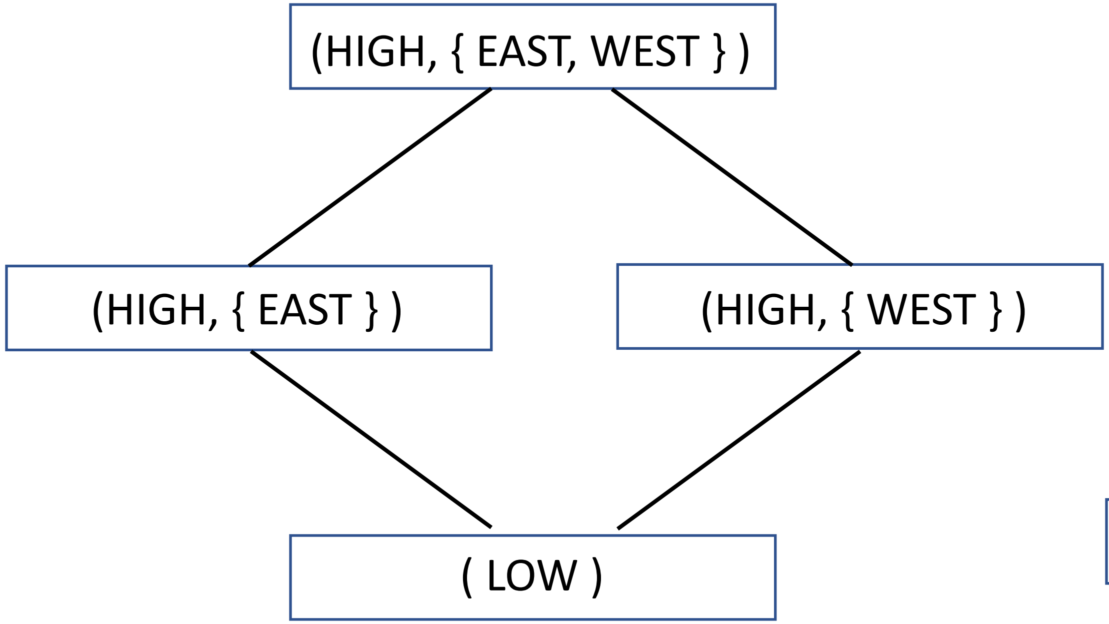


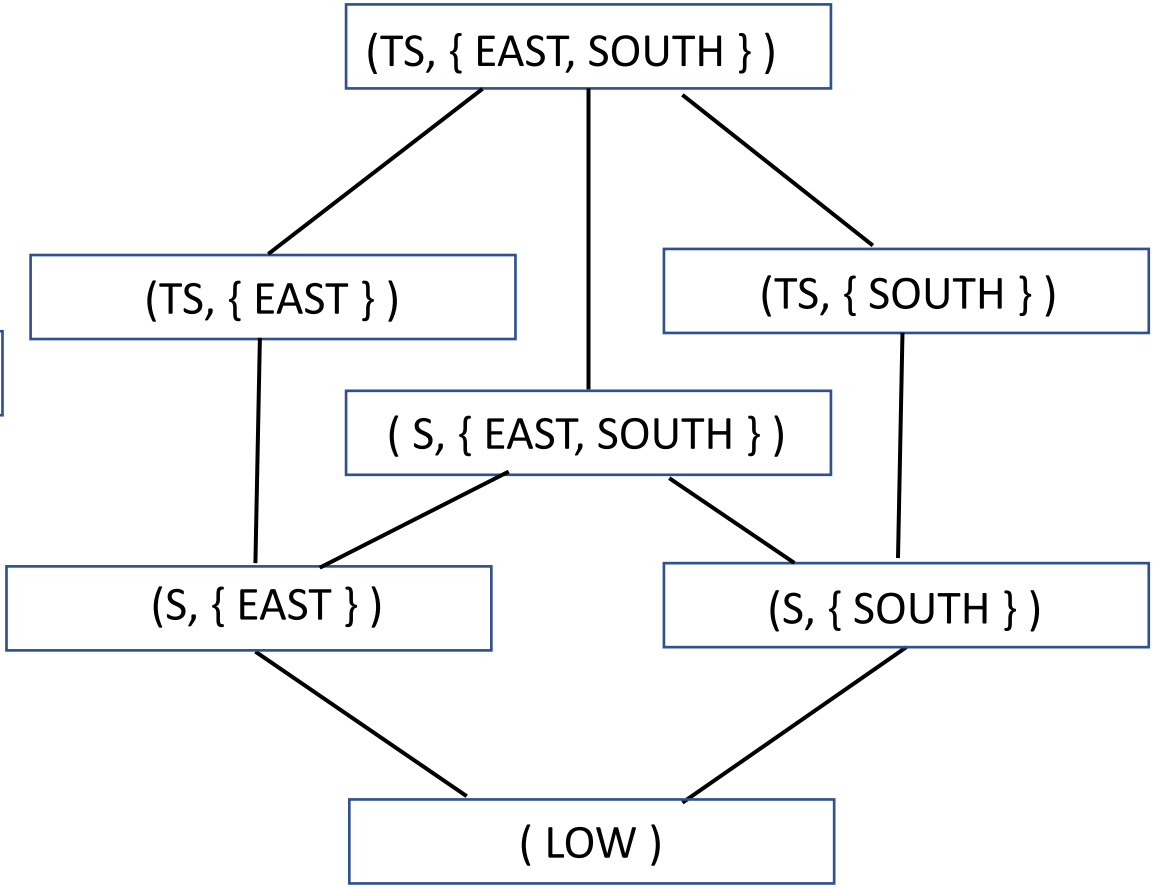
Issues

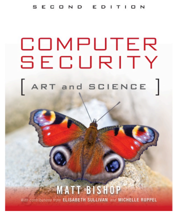
- Compose the lattices
- What is relationship among labels?
  - If the same, trivial
  - If different, new lattice must reflect the relationships among the levels


SECOND EDITION
COMPUTER SECURITY
[ART and SCIENCE]
MATT BISHOP

Example


(HIGH, { EAST, WEST })
(HIGH, { EAST })
(HIGH, { WEST })
(LOW)


(TS, { EAST, SOUTH })
(TS, { EAST })
(TS, { SOUTH })
(S, { EAST, SOUTH })
(S, { EAST })
(S, { SOUTH })
(LOW)

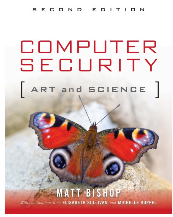


Analysis

- Assume S < HIGH < TS
- Assume SOUTH, EAST, WEST different
- Resulting lattice has:
  - 4 clearances (LOW < S < HIGH < TS)
  - 3 categories (SOUTH, EAST, WEST)

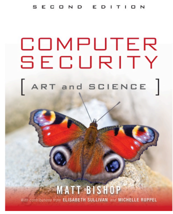
Same Policies

- If we can change policies that components must meet, composition is trivial (as above)
- If we *cannot*, we must show composition meets the same policy as that of components; this can be very hard



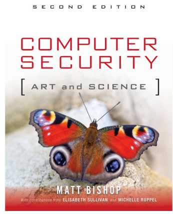
Different Policies

- What does "secure" now mean?
- Which policy (components) dominates?
- Possible principles:
  - Any access allowed by policy of a component must be allowed by composition of components (*autonomy*)
  - Any access forbidden by policy of a component must be forbidden by composition of components (*security*)



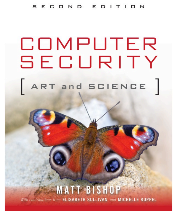
Implications

- Composite system satisfies security policy of components as components' policies take precedence
- If something neither allowed nor forbidden by principles, then:
  - Allow it (Gong & Qian)
  - Disallow it (Fail-Safe Defaults)



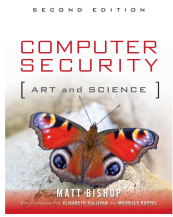
Example

- System X: Bob can’t access Alice’s files
- System Y: Eve, Lilith can access each other’s files
- Composition policy:
  - Bob can access Eve’s files
  - Lilith can access Alice’s files
- Question: can Bob access Lilith’s files?



Solution (Gong & Qian)

- Notation:
  - (*a*, *b*): *a* can read *b*'s files
  - AS(*x*): access set of system *x*
- Set-up:
  - AS(X) = $\varnothing$
  - AS(Y) = { (Eve, Lilith), (Lilith, Eve) }
  - AS(X$\cup$Y) = { (Bob, Eve), (Lilith, Alice), (Eve, Lilith), (Lilith, Eve) }

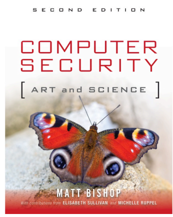


Solution (Gong & Qian)

- Compute transitive closure of $AS(X \cup Y)$:
  - $AS(X \cup Y)^+$ = { (Bob, Eve), (Bob, Lilith), (Bob, Alice), (Eve, Lilith), (Eve, Alice), (Lilith, Eve), (Lilith, Alice) }
- Delete accesses conflicting with policies of components:
  - Delete (Bob, Alice)
- (Bob, Lilith) in set, so Bob can access Lilith's files

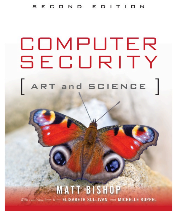
Idea

- Composition of policies allows accesses not mentioned by original policies
- Generate all possible allowed accesses
  - Computation of transitive closure
- Eliminate forbidden accesses
  - Removal of accesses disallowed by individual access policies
- Everything else is allowed
- Note: determining if access allowed is of polynomial complexity



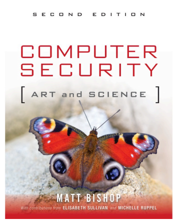
Interference

- Think of it as something used in communication
  - Holly/Lara example: Holly interferes with the CPU utilization, and Lara detects it — communication
- Plays role of writing (interfering) and reading (detecting the interference)



Model

- System as state machine
  - Subjects $S = \{ s_i \}$
  - States $\Sigma = \{ \sigma_i \}$
  - Outputs $O = \{ o_i \}$
  - Commands $Z = \{ z_i \}$
  - State transition commands $C = S \times Z$
- Note: no inputs
  - Encode either as selection of commands or in state transition commands

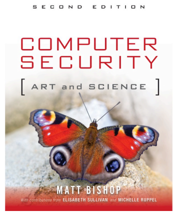


Functions

- State transition function T: $C \times \Sigma \rightarrow \Sigma$
  - Describes effect of executing command c in state σ
- Output function P: $C \times \Sigma \rightarrow O$
  - Output of machine when executing command c in state σ
- Initial state is σ_0

Example: 2-Bit Machine

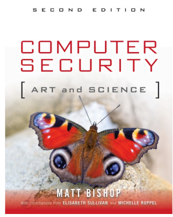
- Users Heidi (high), Lucy (low)
- 2 bits of state, *H* (high) and *L* (low)
  - System state is (*H*, *L*) where *H*, *L* are 0, 1
- 2 commands: *xor0*, *xor1* do xor with 0, 1
  - Operations affect *both* state bits regardless of whether Heidi or Lucy issues it



Example: 2-bit Machine

- S = { Heidi, Lucy }
- Σ = { (0,0), (0,1), (1,0), (1,1) }
- C = { *xor0*, *xor1* }

	Input States (*H*, *L*)			
	(0,0)	(0,1)	(1,0)	(1,1)
xor0	(0,0)	(0,1)	(1,0)	(1,1)
xor1	(1,1)	(1,0)	(0,1)	(0,0)

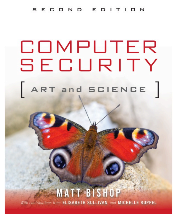


Outputs and States

- T is inductive in first argument, as

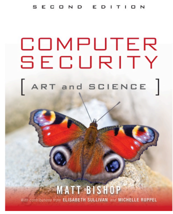
  $T(c_0, \sigma_0) = \sigma_1$; $T(c_{i+1}, \sigma_{i+1}) = T(c_{i+1}, \mathrm{T}(c_i, \sigma_i))$
- Let C^* be set of possible sequences of commands in C
- T^*: $C^* \times \Sigma \rightarrow \Sigma$ and

  $c_s = c_0 \ldots c_n \Rightarrow T^*(c_s, \sigma_i) = T(c_n, \ldots, T(c_0, \sigma_i) \ldots)$
- P similar; define P^*: $C^* \times \Sigma \rightarrow O$ similarly



Projection

- $T^*(c_s,\sigma_i)$ sequence of state transitions
- $P^*(c_s,\sigma_i)$ corresponding outputs
- $proj(s, c_s, \sigma_i)$ set of outputs in $P^*(c_s,\sigma_i)$ that subject s authorized to see
  - In same order as they occur in $P^*(c_s,\sigma_i)$
  - Projection of outputs for s
- Intuition: list of outputs after removing outputs that s cannot see



Purge

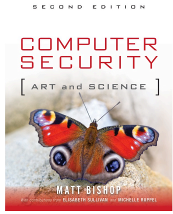
- $G \subseteq S$, G a group of subjects
- $A \subseteq$ Z, A a set of commands
- $\pi_G(c_s)$ subsequence of c_s with all elements (s,z), $s \in G$ deleted
- $\pi_A(c_s)$ subsequence of c_s with all elements (s,z), $z \in A$ deleted
- $\pi_{G,A}(c_s)$ subsequence of c_s with all elements (s,z), $s \in G$ and z $\in A$ deleted

Example: 2-bit Machine

- Let $\sigma_0 = (0,1)$
- 3 commands applied:
  - Heidi applies *xor0*
  - Lucy applies *xor1*
  - Heidi applies *xor1*
- c_s = ((Heidi, *xor0*), (Lucy, *xor1*), (Heidi, *xor1*))
- Output is 011001
  - Shorthand for sequence (0,1) (1,0) (0,1)

Example

- $proj(\text{Heidi}, c_s, \sigma_0) = 011001$
- $proj(\text{Lucy}, c_s, \sigma_0) = 101$
- $\pi_{\text{Lucy}}(c_s) = (\text{Heidi}, \mathit{xor0}), (\text{Heidi}, \mathit{xor1})$
- $\pi_{\text{Lucy},\mathit{xor1}}(c_s) = (\text{Heidi}, \mathit{xor0}), (\text{Heidi}, \mathit{xor1})$
- $\pi_{\text{Heidi}}(c_s) = (\text{Lucy}, \mathit{xor1})$
- $\pi_{\text{Lucy},\mathit{xor0}}(c_s) = (\text{Heidi}, \mathit{xor0}), (\text{Lucy}, \mathit{xor1}), (\text{Heidi}, \mathit{xor1})$
- $\pi_{\text{Heidi},\mathit{xor0}}(c_s) = \pi_{\mathit{xor0}}(c_s) = (\text{Lucy}, \mathit{xor1}), (\text{Heidi}, \mathit{xor1})$
- $\pi_{\text{Heidi},\mathit{xor1}}(c_s) = (\text{Heidi}, \mathit{xor0}), (\text{Lucy}, \mathit{xor1})$
- $\pi_{\mathit{xor1}}(c_s) = (\text{Heidi}, \mathit{xor0})$

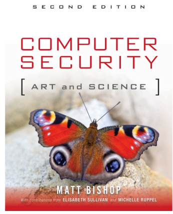


Noninterference

- Intuition: If set of outputs Lucy can see corresponds to set of inputs she can see, there is no interference
- Formally: G, $G' \subseteq$ S, $G \neq G'$; $A \subseteq Z$; users in G executing commands in A are *noninterfering* with users in G' iff for all $c_s \in C^*$, and for all $s \in G'$,

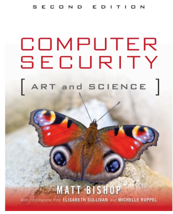
$$proj(s, c_s, \sigma_i) = proj(s, \pi_{G,A}(c_s), \sigma_i)$$

  - Written A,G :| G'



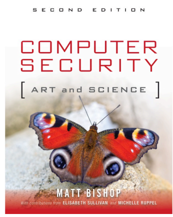
Example: 2-Bit Machine

- Let c_s = ((Heidi, *xor0*), (Lucy, *xor1*), (Heidi, *xor1*)) and $\sigma_0 = (0, 1)$
  - As before
- Take G = { Heidi }, G' = { Lucy }, $A = \varnothing$
- $\pi_{\text{Heidi}}(c_s)$ = (Lucy, *xor1*)
  - So *proj*(Lucy, $\pi_{\text{Heidi}}(c_s)$, σ_0) = 0
- *proj*(Lucy, c_s, σ_0) = 101
- So { Heidi } :| { Lucy } is false
  - Makes sense; commands issued to change *H* bit also affect *L* bit



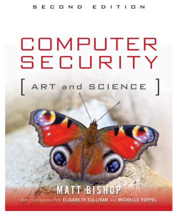
Example

- Same as before, but Heidi's commands affect *H* bit only, Lucy's the *L* bit only
- Output is $0_H 0_L 1_H$
- $\pi_{\text{Heidi}}(c_s)$ = (Lucy, *xor1*)
  - So $proj(\text{Lucy}, \pi_{\text{Heidi}}(c_s), \sigma_0) = 0$
- $proj(\text{Lucy}, c_s, \sigma_0) = 0$
- So { Heidi } :| { Lucy } is true
  - Makes sense; commands issued to change *H* bit now do not affect *L* bit



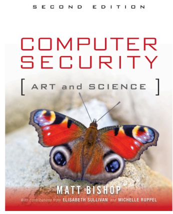
Security Policy

- Partitions systems into authorized, unauthorized states
- Authorized states have no forbidden interferences
- Hence a *security policy* is a set of noninterference assertions
    - See previous definition



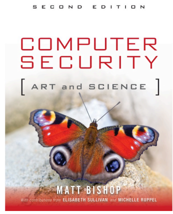
Alternative Development

- System X is a set of protection domains $D = \{ d_1, \ldots, d_n \}$
- When command c executed, it is executed in protection domain $dom(c)$
- Give alternate versions of definitions shown previously



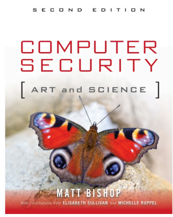
Security Policy

- $D = \{ d_1, \ldots, d_n \}$, d_i a protection domain
- r: $D \times D$ a reflexive relation
- Then r defines a security policy
- Intuition: defines how information can flow around a system
  - $d_i r d_j$ means info can flow from d_i to d_j
  - $d_i r d_i$ as info can flow within a domain



Projection Function

- π' analogue of π, earlier
- Commands, subjects absorbed into protection domains
- $d \in D$, $c \in C$, $c_s \in C^*$
- $\pi'_d(\nu) = \nu$
- $\pi'_d(c_s c) = \pi'_d(c_s)c$ if $dom(c) r d$
- $\pi'_d(c_s c) = \pi'_d(c_s)$ otherwise
- Intuition: if executing *c* interferes with *d*, then *c* is visible; otherwise, as if *c* never executed

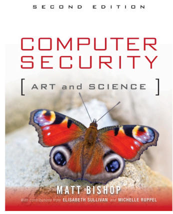


Noninterference-Secure

- System has set of protection domains *D*
- System is *noninterference-secure with respect to policy r* if

$$P^*(c, T^*(c_s, \sigma_0)) = P^*(c, T^*(\pi'_d(c_s), \sigma_0))$$

- Intuition: if executing c_s causes the same transitions for subjects in domain *d* as does its projection with respect to domain *d*, then no information flows in violation of the policy

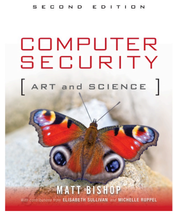


Output-Consistency

- $c \in C$, $dom(c) \in D$
- $\sim^{dom(c)}$ equivalence relation on states of system X
- $\sim^{dom(c)}$ *output-consistent* if

$$\sigma_a \sim^{dom(c)} \sigma_b \Rightarrow P(c, \sigma_a) = P(c, \sigma_b)$$

- Intuition: states are output-consistent if for subjects in *dom*(*c*), projections of outputs for both states after *c* are the same



Lemma

- Let $T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0)$ for $c \in C$
- If $\sim^d$ output-consistent, then system is noninterference-secure with respect to policy r

Proof

- $d = dom(c)$ for $c \in C$
- By definition of output-consistent,

$$T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0)$$

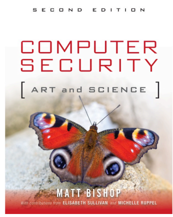
implies

$$P^*(c, T^*(c_s, \sigma_0)) = P^*(c, T^*(\pi'_d(c_s), \sigma_0))$$

- This is definition of noninterference-secure with respect to policy r

Unwinding Theorem

- Links security of sequences of state transition commands to security of individual state transition commands
- Allows you to show a system design is multilevel-secure by showing it matches specs from which certain lemmata derived
  - Says *nothing* about security of system, because of implementation, operation, *etc*. issues

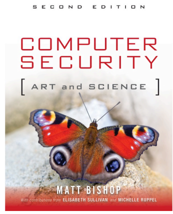


Locally Respects

- *r* is a policy
- System *X* *locally respects* *r* if *dom*(*c*) being noninterfering with $d \in D$ implies $\sigma_a \sim^d T(c, \sigma_a)$
- Intuition: when *X* locally respects *r*, applying *c* under policy *r* to system *X* has no effect on domain *d*

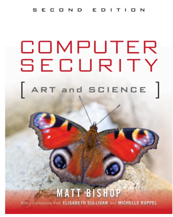
Transition-Consistent

- r policy, $d \in D$
- If $\sigma_a \sim^d \sigma_b$ implies $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, system X is *transition-consistent* under r
- Intuition: command c does not affect equivalence of states under policy r



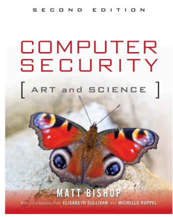
Unwinding Theorem

- Links security of sequences of state transition commands to security of individual state transition commands
- Allows you to show a system design is ML secure by showing it matches specs from which certain lemmata derived
  - Says *nothing* about security of system, because of implementation, operation, *etc*. issues



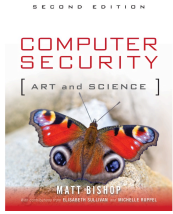
Locally Respects

- *r* is a policy
- System *X* locally respects *r* if *dom*(*c*) being noninterfering with $d \in D$ implies $\sigma_a \sim^d T(c, \sigma_a)$
- Intuition: applying *c* under policy *r* to system *X* has no effect on domain *d* when *X* locally respects *r*



Transition-Consistent

- r policy, d $\in D$
- If $\sigma_a \sim^d \sigma_b$ implies $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, system X transition-consistent under r
- Intuition: command c does not affect equivalence of states under policy r



Theorem

- *r* policy, *X* system that is output consistent, transition consistent, and locally respects *r*
- Then *X* noninterference-secure with respect to policy *r*
- Significance: basis for analyzing systems claiming to enforce noninterference policy
  - Establish conditions of theorem for particular set of commands, states with respect to some policy, set of protection domains
  - Noninterference security with respect to *r* follows

Proof

Must show $\sigma_a \sim^d \sigma_b \Rightarrow T^*(c_s, \sigma_a) \sim^d T^*(\pi'_d(c_s), \sigma_b)$

- Induct on length of c_s
- Basis: if $c_s = \nu$, then $T^*(c_s, \sigma_a) = \sigma_a$ and $\pi'_d(\nu) = \nu$; claim holds
- Hypothesis: for $c_s = c_1 \dots c_n$, $\sigma_a \sim^d \sigma_b \Rightarrow T^*(c_s, \sigma_a) \sim^d T^*(\pi'_d(c_s), \sigma_b)$

Induction Step

- Consider $c_s c_{n+1}$. Assume $\sigma_a \sim^d \sigma_b$ and look at $T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$
- 2 cases:
  - $dom(c_{n+1})rd$ holds
  - $dom(c_{n+1})rd$ does not hold

$dom(c_{n+1})rd$ Holds

$T^*(\pi'_d(c_s c_{n+1}), \sigma_b) = T^*(\pi'_d(c_s)c_{n+1}, \sigma_b) = T(c_{n+1}, T^*(\pi'_d(c_s), \sigma_b))$

- By definition of T^* and π'_d

$\sigma_a \sim^d \sigma_b \Rightarrow T(c_{n+1}, \sigma_a) \sim^d T(c_{n+1}, \sigma_b)$

- As X transition-consistent

$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s), \sigma_b))$

- By transition-consistency and IH

$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$

- By substitution from earlier equality

$T^*(c_s c_{n+1}, \sigma_a) \sim^d T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$

- By definition of T^*

proving hypothesis

$dom(c_{n+1})rd$ Does Not Hold

$T^*(\pi'_d(c_s c_{n+1}), \sigma_b) = T^*(\pi'_d(c_s), \sigma_b)$

- By definition of π'_d

$T^*(c_s, \sigma_a) = T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$

- By above and IH

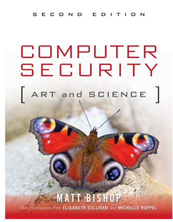
$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(c_s, \sigma_a)$

- As X locally respects r, $\sigma \sim^d T(c_{n+1}, \sigma)$ for any σ

$T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$

- Substituting back

proving hypothesis

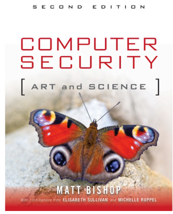


Finishing Proof

- Take $\sigma_a = \sigma_b = \sigma_0$, so from claim proved by induction,

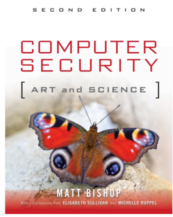
$$T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0)$$

- By previous lemma, as *X* (and so $\sim^d$) output consistent, then *X* is noninterference-secure with respect to policy *r*



Access Control Matrix

- Example of interpretation
- Given: access control information
- Question: are given conditions enough to provide noninterference security?
- Assume: system in a particular state
  - Encapsulates values in ACM

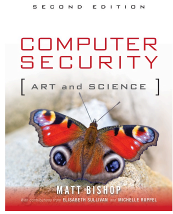


ACM Model

- Objects $L = \{ l_1, \ldots, l_m \}$
  - Locations in memory
- Values $V = \{ v_1, \ldots, v_n \}$
  - Values that L can assume
- Set of states $\Sigma = \{ \sigma_1, \ldots, \sigma_k \}$
- Set of protection domains $D = \{ d_1, \ldots, d_j \}$

Functions

- *value*: $L \times \Sigma \to V$
  - returns value *v* stored in location *l* when system in state σ
- *read*: $D \to 2^V$
  - returns set of objects observable from domain *d*
- *write*: $D \to 2^V$
  - returns set of objects observable from domain *d*

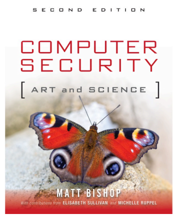


Interpretation of ACM

- Functions represent ACM
  - Subject *s* in domain *d*, object *o*
  - $r \in A[s, o]$ if $o \in read(d)$
  - $w \in A[s, o]$ if $o \in write(d)$
- Equivalence relation:

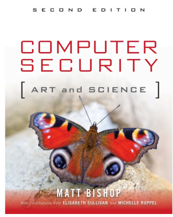
$$[\sigma_a \sim^{dom(c)} \sigma_b] \Leftrightarrow [\ \forall l_i \in read(d)\ [\ value(l_i, \sigma_a) = value(l_i, \sigma_b)\ ]\ ]$$

  - You can read the *exactly* the same locations in both states



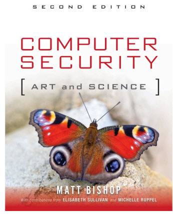
Enforcing Policy *r*

- 5 requirements
  - 3 general ones describing dependence of commands on rights over input and output
    - Hold for all ACMs and policies
  - 2 that are specific to some security policies
    - Hold for *most* policies



Enforcing Policy *r*: General Requirements

- Output of command *c* executed in domain *dom*(*c*) depends only on values for which subjects in *dom*(*c*) have read access
  - $\sigma_a \sim^{dom(c)} \sigma_b \Rightarrow P(c, \sigma_a) = P(c, \sigma_b)$
- If *c* changes l_i, then *c* can only use values of objects in *read*(*dom*(*c*)) to determine new value
  - $[\ \sigma_a \sim^{dom(c)} \sigma_b \wedge$
    $(value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a) \vee value(l_i, T(c, \sigma_b)) \neq value(l_i, \sigma_b))\ ] \Rightarrow$
    $value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$
- If *c* changes l_i, then *dom*(*c*) provides subject executing *c* with write access to l_i
  - $value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a) \Rightarrow l_i \in write(dom(c))$



Enforcing Policies *r*: Specific to Policy

- If domain u can interfere with domain v, then every object that can be read in u can also be read in v; so if object o cannot be read in u, but can be read in v and object o' in u can be read in v, then info flows from o to o', then to v

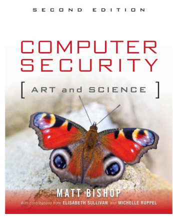
$$[\ u, v \in D \wedge urv\ ] \Rightarrow read(u) \subseteq read(v)$$

- Subject s can write object o in v, subject s' can read o in u, then domain v can interfere with domain u

$$[\ l_i \in read(u) \wedge l_i \in write(v)\ ] \Rightarrow vru$$

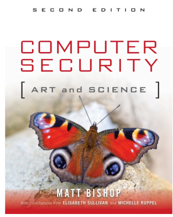
Theorem

- Let *X* be a system satisfying these five conditions. Then *X* is noninterference-secure with respect to r
- Proof: must show *X* output-consistent, locally respects *r*, transition-consistent
  - Then by unwinding theorem, this theorem holds



Output-Consistent

- Take equivalence relation to be $\sim^{d}$, first condition *is* definition of output-consistent

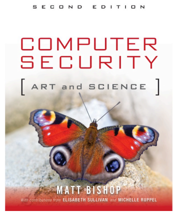


Locally Respects *r*

- Proof by contradiction: assume $(dom(c), d) \notin r$ but $\sigma_a \sim^d T(c, \sigma_a)$ does not hold
- Some object has value changed by c:

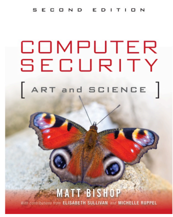
$$\exists\, l_i \in read(d)\ [\ value(l_i, \sigma_a) \neq \text{value}(l_i, T(c, \sigma_a))\ ]$$

- Condition 3: $l_i \in write(d)$
- Condition 5: $dom(c) r d$, contradiction
- So $\sigma_a \sim^d T(c, \sigma_a)$ holds, meaning X locally respects r



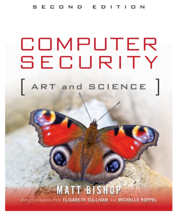
Transition Consistency

- Assume $\sigma_a \sim^d \sigma_b$
- Must show $value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$ for $l_i \in read(d)$
- 3 cases dealing with change that c makes in l_i in states σ_a, σ_b
  - $value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a)$
  - $value(l_i, T(c, \sigma_b)) \neq value(l_i, \sigma_b)$
  - Neither of the above two hold



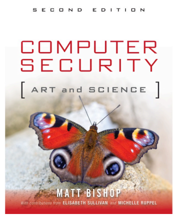
Case 1: $value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a)$

- Condition 3: $l_i \in write(dom(c))$
- As $l_i \in read(d)$, condition 5 says $dom(c) r d$
- Condition 4: $read(dom(c)) \subseteq read(d)$
- As $\sigma_a \sim^d \sigma_b$, $\sigma_a \sim^{dom(c)} \sigma_b$
- Condition 2: $value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$
- So $T(c, \sigma_a) \sim^{dom(c)} T(c, \sigma_b)$, as desired



Case 2: $value(l_i, T(c, \sigma_b)) \neq value(l_i, \sigma_b)$

- Condition 3: $l_i \in write(dom(c))$
- As $l_i \in read(d)$, condition 5 says $dom(c) rd$
- Condition 4: $read(dom(c)) \subseteq read(d)$
- As $\sigma_\mathrm{a} \sim^{d} \sigma_\mathrm{b}$, $\sigma_\mathrm{a} \sim^{dom(c)} \sigma_\mathrm{b}$
- Condition 2: $value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$
- So $T(c, \sigma_a) \sim^{dom(c)} T(c, \sigma_b)$, as desired



Case 3: Neither of the Previous Two Hold

- This means the two conditions below hold:
  - $value(l_i, T(c, \sigma_a)) = value(l_i, \sigma_a)$
  - $value(l_i, T(\mathrm{c}, \sigma_b)) = value(l_i, \sigma_b)$
- Interpretation of $\sigma_a \sim^d \sigma_b$ is:

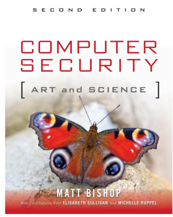
$$\text{for } l_i \in read(d),\ value(l_i, \sigma_a) = value(l_i, \sigma_b)$$

- So $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, as desired

In all 3 cases, *X* transition-consistent

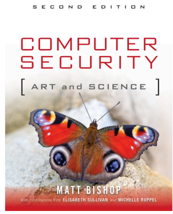
Policies Changing Over Time

- Problem: previous analysis assumes static system
  - In real life, ACM changes as system commands issued
- Example: $w \in C^*$ leads to current state
  - *cando*(w, s, z) holds if s can execute z in current state
  - Condition noninterference on *cando*
  - If $\neg$*cando*(w, Lara, "write f"), Lara can't interfere with any other user by writing file f



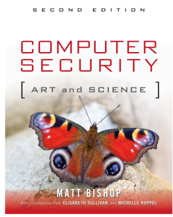
Generalize Noninterference

- $G \subseteq S$ set of subjects, $A \subseteq Z$ set of commands, p predicate over elements of C^*
- $c_s = (c_1, \ldots, c_n) \in C^*$
- $\pi''(\nu) = \nu$
- $\pi''((c_1, \ldots, c_n)) = (c_1', \ldots, c_n')$, where
  - $c_i' = \nu$ if $p(c_1', \ldots, c_{i-1}')$ and $c_i = (s, z)$ with $s \in G$ and $z \in A$
  - $c_i' = c_i$ otherwise



Intuition

- $\pi''(c_s) = c_s$
- But if p holds, and element of c_s involves both command in A and subject in G, replace corresponding element of c_s with empty command ν
    - Just like deleting entries from c_s as $\pi_{A,G}$ does earlier



Noninterference

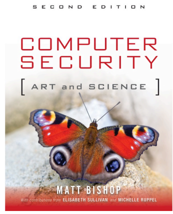
- G, $G' \subseteq S$ sets of subjects, $A \subseteq Z$ set of commands, p predicate over C^*
- Users in G executing commands in A are *noninterfering with users in* G' under condition p iff, for all $c_s \in C^*$ and for all $s \in G'$, $proj(s, c_s, \sigma_i) = proj(s, \pi''(c_s), \sigma_i)$
    - Written A,G :| G' **if** p

Example

- From earlier one, simple security policy based on noninterference:

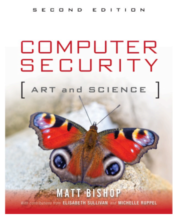
$$\forall(s \in S)\ \forall(z \in Z)\ [\ \{z\}, \{s\} :|\ S \textbf{ if } \neg cando(w, s, z)\ ]$$

- If subject can't execute command (the $\neg$*cando* part) in any state, subject can't use that command to interfere with another subject



Another Example

- Consider system in which rights can be passed
  - *pass*(s, z) gives s right to execute z
  - $w_n = v_1, \ldots, v_n$ sequence of $v_i \in C^*$
  - $prev(w_n) = w_{n-1}$; $last(w_n) = v_n$

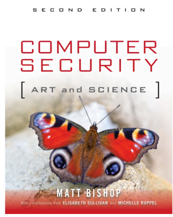


Policy

- No subject *s* can use *z* to interfere if, in previous state, *s* did not have right to *z*, and no subject gave it to *s*

  $\{ z \}, \{ s \} :| S$

$$\textbf{if } [\, \neg cando(prev(w), s, z) \land [\, cando(prev(w), s', pass(s, z)) \Rightarrow \neg last(w) = (s', pass(s, z)) \,]\,]$$



Effect

- Suppose $s_1 \in S$ can execute *pass*(s_2, z)
- For all $w \in C^*$, *cando*(w, s_1, *pass*(s_2, z)) holds
- Initially, *cando*(ν, s_2, z) false
- Let $z' \in Z$ be such that (s_3, z') noninterfering with (s_2, z)
    - So for each w_n with $v_n = (s_3, z')$, *cando*(w_n, s_2, z) = *cando*(w_{n-1}, s_2, z)

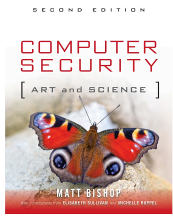
Effect

- Then policy says for all $s \in S$

  $proj(s, ((s_2, z), (s_1, pass(s_2, z)), (s_3, z'), (s_2, z)), \sigma_i) =$

  $\qquad proj(s, ((s_1, pass(s_2, z)), (s_3, z'), (s_2, z)), \sigma_i)$

- So s_2's first execution of z does not affect any subject's observation of system

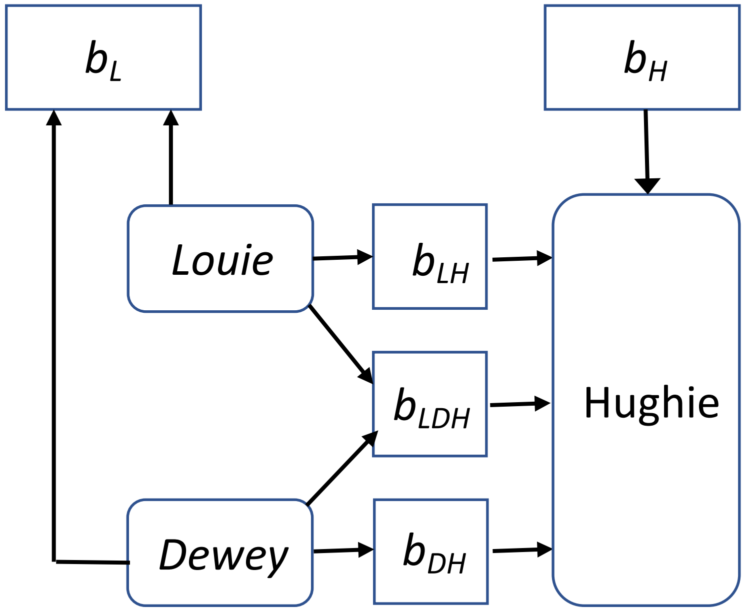


Policy Composition I

- Assumed: Output function of input
  - Means deterministic (else not function)
  - Means uninterruptability (differences in timings can cause differences in states, hence in outputs)
- This result for deterministic, noninterference-secure systems

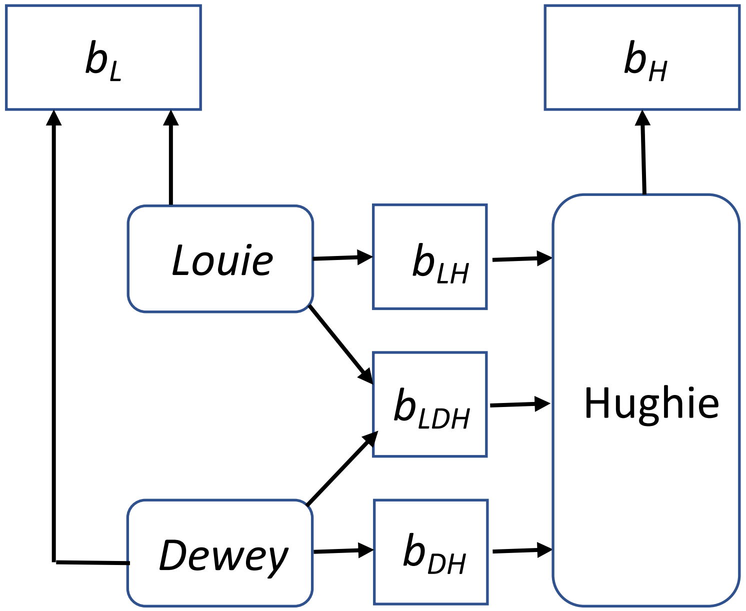
Compose Systems

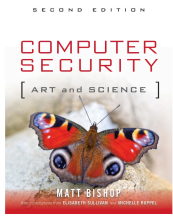
- Louie, Dewey LOW
- Hughie HIGH
- b_L output buffer
  - Anyone can read it
- b_H input buffer
  - From HIGH source
- Hughie reads from:
  - b_{LH} (Louie writes)
  - b_{LDH} (Louie, Dewey write)
  - b_{DH} (Dewey writes)



Systems Secure

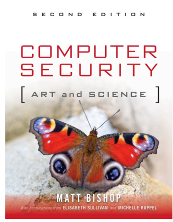
- All noninterference-secure
  - Hughie has no output
    - So inputs don't interfere with it
  - Louie, Dewey have no input
    - So (nonexistent) inputs don't interfere with outputs





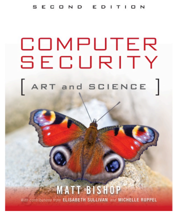
Security of Composition

- Buffers finite, sends/receives blocking: composition *not* secure!
  - Example: assume b_{DH}, b_{LH} have capacity 1
- Algorithm:
  1. Louie (Dewey) sends message to b_{LH} (b_{DH})
     – Fills buffer
  2. Louie (Dewey) sends second message to b_{LH} (b_{DH})
  3. Louie (Dewey) sends a 0 (1) to b_L
  4. Louie (Dewey) sends message to b_{LDH}
     – Signals Hughie that Louie (Dewey) completed a cycle



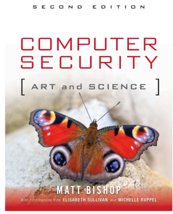
Hughie

- Reads bit from b_H
  - If 0, receive message from b_{LH}
  - If 1, receive message from b_{DH}
- Receive on b_{LDH}
  - To wait for buffer to be filled



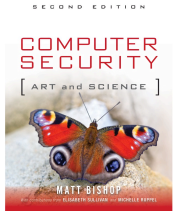
Example

- Hughie reads 0 from b_H
  - Reads message from b_{LH}
- Now Louie's second message goes into b_{LH}
  - Louie completes setp 2 and writes 0 into b_L
- Dewey blocked at step 1
  - Dewey cannot write to b_L
- Symmetric argument shows that Hughie reading 1 produces a 1 in b_L
- So, input from b_H copied to output b_L



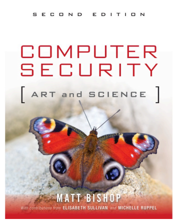
Nondeducibility

- Noninterference: do state transitions caused by high level commands interfere with sequences of state transitions caused by low level commands?
- Really case about inputs and outputs:
  - Can low level subject deduce *anything* about high level outputs from a set of low level outputs?



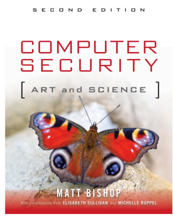
Example: 2-Bit System

- *High* operations change only *High* bit
  - Similar for *Low*
- σ_0 = (0, 0)
- Sequence of commands:
  - (Heidi, *xor1*), (Lara, *xor0*), (Lara, *xor1*), (Lara, *xor0*), (Heidi, *xor1*), (Lara, *xor0*)
  - Both bits output after each command
- Output is: 00101011110101



Security

- Not noninterference-secure w.r.t. Lara
  - Lara sees output as 0001111
  - Delete *High* outputs and she sees 00111
- But Lara still cannot deduce the commands deleted
  - Don’t affect values; only lengths
- So it is deducibly secure
  - Lara can’t deduce the commands Heidi gave



Event System

- 4-tuple (E, I, O, T)
  - E set of events
  - $I \subseteq E$ set of input events
  - $O \subseteq E$ set of output events
  - T set of all finite sequences of events legal within system
- E partitioned into H, L
  - H set of *High* events
  - L set of *Low* events

More Events …

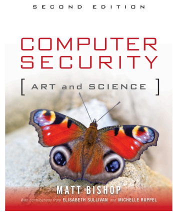
- $H \cap I$ set of *High* inputs
- $H \cap O$ set of *High* outputs
- $L \cap I$ set of *Low* inputs
- $L \cap O$ set of *Low* outputs
- T_{Low} set of all possible sequences of *Low* events that are legal within system
- $\pi_L : T \rightarrow T_{Low}$ projection function deleting all *High* inputs from trace
  - *Low* observer should not be able to deduce anything about *High* inputs from trace $t_{Low} \in T_{low}$

Deducibly Secure

- System deducibly secure if for all traces $t_{Low} \in T_{Low}$, the corresponding set of high level traces contains every possible trace $t \in T$ for which $\pi_L(t) = t_{Low}$
  - Given any t_{Low}, the trace $t \in T$ producing that t_{Low} is equally likely to be *any* trace with $\pi_L(t) = t_{Low}$

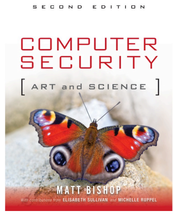
Example: 2-Bit Machine

- Let *xor0*, *xor1* apply to both bits, and both bits output after each command
- Initial state: (0, 1)
- Inputs: $1_H 0_L 1_L 0_H 1_L 0_L$
- Outputs: 10 10 01 01 10 10
- Lara (at *Low*) sees: 001100
  - Does not know initial state, so does not know first input; but can deduce fourth input is 0
- Not deducibly secure



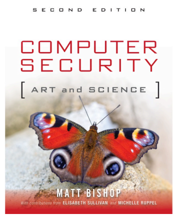
Example: 2-Bit Machine

- Now *xor0*, *xor1* apply only to state bit with same level as user
- Inputs: $1_H 0_L 1_L 0_H 1_L 0_L$
- Outputs: 1011111011
- Lara sees: 01101
- She cannot deduce *anything* about input
  - Could be $0_H 0_L 1_L 0_H 1_L 0_L$ or $0_L 1_H 1_L 0_H 1_L 0_L$ for example
- Deducibly secure



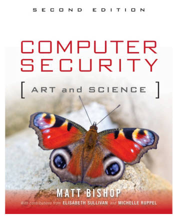
Security of Composition

- In general: deducibly secure systems not composable
- *Strong noninterference*: deducible security + requirement that no *High* output occurs unless caused by a *High* input
  - Systems meeting this property *are* composable



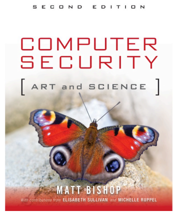
Example

- 2-bit machine done earlier does not exhibit strong noninterference
  - Because it puts out *High* bit even when there is no *High* input
- Modify machine to output only state bit at level of latest input
  - *Now* it exhibits strong noninterference



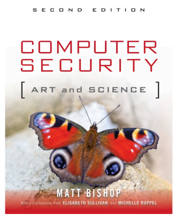
Problem

- Too restrictive; it bans some systems that are *obviously* secure
- Example: System *upgrade* reads *Low* inputs, outputs those bits at *High*
  - Clearly deducibly secure: low level user sees no outputs
  - Clearly does not exhibit strong noninterference, as no high level inputs!



Remove Determinism

- Previous assumption
  - Input, output synchronous
  - Output depends only on commands triggered by input
    - Sometimes absorbed into commands …
  - Input processed one datum at a time
- Not realistic
  - In real systems, lots of asynchronous events



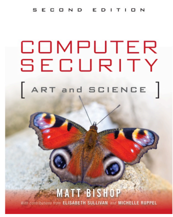
Generalized Noninterference

- Nondeterministic systems meeting noninterference property meet *generalized noninterference-secure property*
    - More robust than nondeducible security because minor changes in assumptions affect whether system is nondeducibly secure

Example

- System with *High* Holly, *Low* Lucy, text file at *High*
  - File fixed size, symbol ✧ marks empty space
  - Holly can edit file, Lucy can run this program:

```
while true do begin
      n := read_integer_from_user;
      if n > file_length or char_in_file[n] = ✧ then
              print random_character;
      else
              print char_in_file[n];
end;
```



Security of System

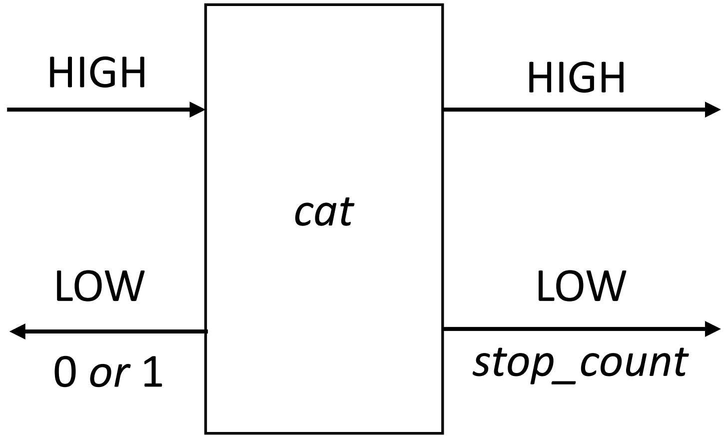
- Not noninterference-secure
  - High level inputs—Holly’s changes—affect low level outputs
- *May* be deducibly secure
  - Can Lucy deduce contents of file from program?
  - If output meaningful (“This is right”) or close (“Thes is riqht”), yes
  - Otherwise, no
- So deducibly secure depends on which inferences are allowed

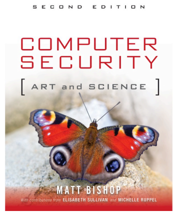
Composition of Systems

- Does composing systems meeting generalized noninterference-secure property give you a system that also meets this property?
- Define two systems (*cat*, *dog*)
- Compose them

First System: *cat*

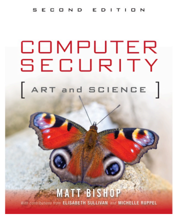
- Inputs, outputs can go left or right
- After some number of inputs, *cat* sends two outputs
  - First *stop_count*
  - Second parity of *High* inputs, outputs





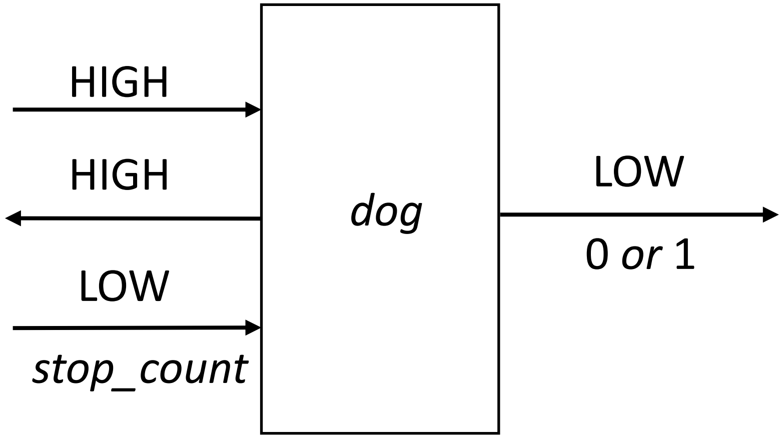
Noninterference-Secure?

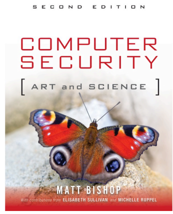
- If even number of *High* inputs, output could be:
  - 0 (even number of outputs)
  - 1 (odd number of outputs)
- If odd number of *High* inputs, output could be:
  - 0 (odd number of outputs)
  - 1 (even number of outputs)
- High level inputs do not affect output
  - So noninterference-secure



Second System: *dog*

- High outputs to left
- Low outputs of 0 or 1 to right
- *stop_count* input from the left
    - When it arrives, *dog* emits 0 or 1

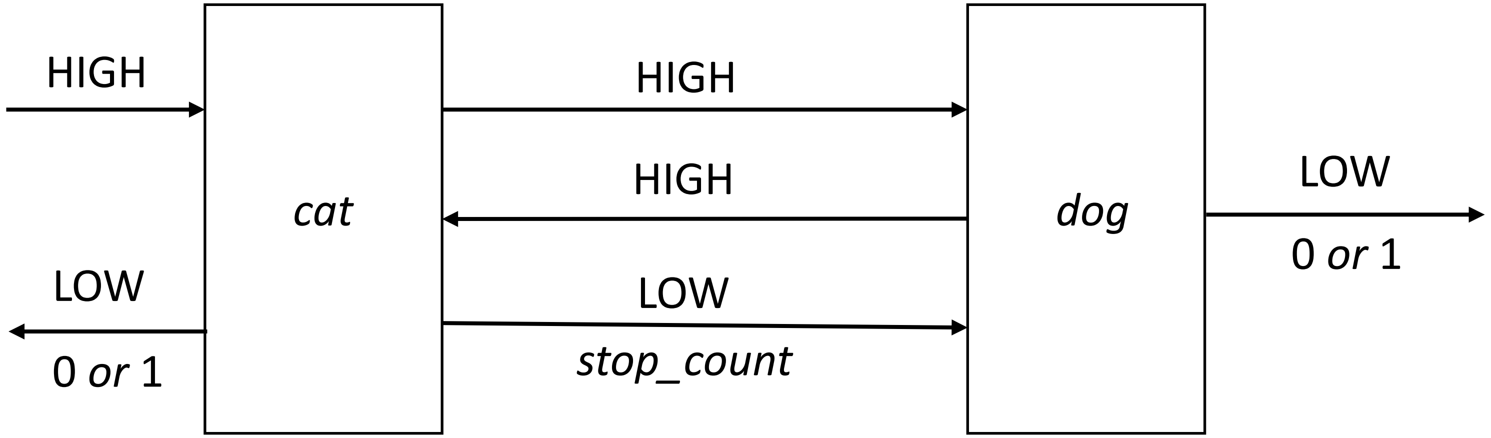




Noninterference-Secure?

- When *stop_count* arrives:
  - May or may not be inputs for which there are no corresponding outputs
  - Parity of *High* inputs, outputs can be odd or even
  - Hence *dog* emits 0 or 1
- High level inputs do not affect low level outputs
  - So noninterference-secure

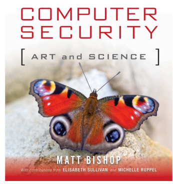
Compose Them



- Once sent, message arrives
  - But *stop_count* may arrive before all inputs have generated corresponding outputs
  - If so, even number of *High* inputs and outputs on *cat*, but odd number on *dog*
- Four cases arise

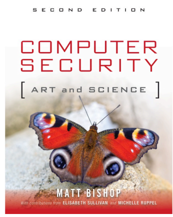
The Cases

- *cat*, odd number of inputs, outputs; *dog*, even number of inputs, odd number of outputs
  - Input message from *cat* not arrived at *dog*, contradicting assumption
- *cat*, even number of inputs, outputs; *dog*, odd number of inputs, even number of outputs
  - Input message from *dog* not arrived at *cat*, contradicting assumption



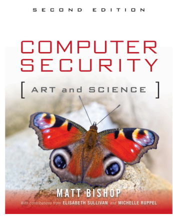
The Cases

- cat, odd number of inputs, outputs; dog, odd number of inputs, even number of outputs
  - dog sent even number of outputs to cat, so cat has had at least one input from left
- cat, even number of inputs, outputs; dog, even number of inputs, odd number of outputs
  - dog sent odd number of outputs to cat, so cat has had at least one input from left



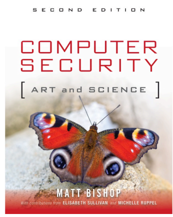
The Conclusion

- Composite system *catdog* emits 0 to left, 1 to right (or 1 to left, 0 to right)
  - Must have received at least one input from left
- Composite system *catdog* emits 0 to left, 0 to right (or 1 to left, 1 to right)
  - Could not have received any from left (i.e., no HIGH inputs)
- So, *High* inputs affect *Low* outputs
  - Not noninterference-secure



Feedback-Free Systems

- System has *n* distinct components
- Components c_i, c_j are *connected* if any output of c_i is input to c_j
- System is *feedback-free* if for all c_i connected to c_j, c_j not connected to any c_i
  - Intuition: once information flows from one component to another, no information flows back from the second to the first



Feedback-Free Security

- *Theorem*: A feedback-free system composed of noninterference-secure systems is itself noninterference-secure

Some Feedback

- *Lemma*: A noninterference-secure system can feed a HIGH output *o* to a HIGH input *i* if the arrival of *o* at the input of the next component is delayed until *after* the next LOW input or output
- *Theorem*: A system with feedback as described in the above lemma and composed of noninterference-secure systems is itself noninterference-secure

Why Didn’t They Work?

- For compositions to work, machine must act same way regardless of what precedes LOW input (HIGH, LOW, nothing)
- *dog* does not meet this criterion
  - If first input is *stop_count*, *dog* emits 0
  - If high level input precedes *stop_count*, *dog* emits 0 or 1

State Machine Model: 2-Bit Machine

Levels *High*, *Low*, meet 4 properties:

1. For every input i_k, state σ_j, there is an element $c_m \in C^*$ such that $T^*(c_m, \sigma_j) = \sigma_n$, where $\sigma_n \neq \sigma_j$

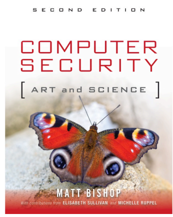
T^* is total function, inputs and commands always move system to a different state

Property 2

2. There is an equivalence relation $\equiv$ such that:
   a. If system in state σ_i and HIGH sequence of inputs causes transition from σ_i to σ_j, then $\sigma_i \equiv \sigma_j$
      - 2 states equivalent if either reachable from the other state using only HIGH commands
   b. If $\sigma_i \equiv \sigma_j$ and LOW sequence of inputs $i_1, \ldots, i_n$ causes system in state σ_i to transition to σ_i', then there is a state σ_j' such that $\sigma_i' \equiv \sigma_j'$ and inputs $i_1, \ldots, i_n$ cause system in state σ_j to transition to σ_j'
      - States resulting from giving same LOW commands to the two equivalent original states have same LOW projection

$\equiv$ holds if LOW projections of both states are same

- If 2 states equivalent, HIGH commands do not affect LOW projections



Property 3

- Let $\sigma_i \equiv \sigma_j$. If sequence of HIGH outputs $o_1, \ldots, o_n$ indicate system in state σ_i transitioned to state σ_i', then for some state σ_j' with $\sigma_j' \equiv \sigma_i'$, sequence of HIGH outputs $o_1', \ldots, o_m'$ indicates system in σ_j transitioned to σ_j'
  - HIGH outputs do not indicate changes in LOW projection of states

Property 4

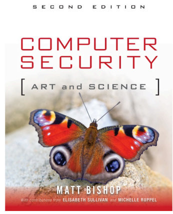
- Let $\sigma_i \equiv \sigma_j$, let c, d be HIGH output sequences, e a LOW output. If output sequence ced indicates system in state σ_i transitions to σ_i', then there are HIGH output sequences c' and d' and state σ_j' such that $c'ed'$ indicates system in state σ_j transitions to state σ_j'
  - Intermingled LOW, HIGH outputs cause changes in LOW state reflecting LOW outputs only

Restrictiveness

- System is *restrictive* if it meets the preceding 4 properties

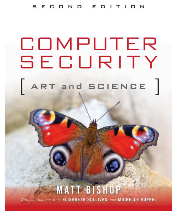
Composition

- Intuition: by 3 and 4, HIGH output followed by LOW output has same effect as the LOW input, so composition of restrictive systems should be restrictive



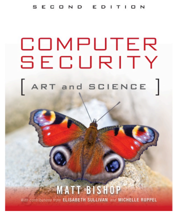
Composite System

- System M_1's outputs are acceptable as M_2's inputs
- μ_{1i}, μ_{2i} states of M_1, M_2
- States of composite system pairs of M_1, M_2 states (μ_{1i}, μ_{2i})
- *e* event causing transition
- *e* causes transition from state (μ_{1a}, μ_{2a}) to state (μ_{1b}, μ_{2b}) if any of 3 conditions hold



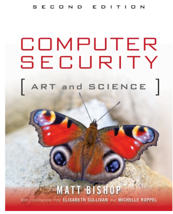
Conditions

1. M_1 in state μ_{1a} and *e* occurs, M_1 transitions to μ_{1b}; *e* not an event for M_2; and $\mu_{2a} = \mu_{2b}$
2. M_2 in state μ_{2a} and *e* occurs, M_2 transitions to μ_{2b}; *e* not an event for M_1; and $\mu_{1a} = \mu_{1b}$
3. M_1 in state μ_{1a} and *e* occurs, M_1 transitions to μ_{1b}; M_2 in state μ_{2a} and *e* occurs, M_2 transitions to μ_{2b}; *e* is input to one machine, and output from other



Intuition

- Event causing transition in composite system causes transition in at least 1 of the components
- If transition occurs in exactly 1 component, event must not cause transition in other component when not connected to the composite system

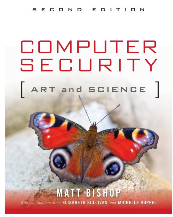


Equivalence for Composite

- Equivalence relation for composite system

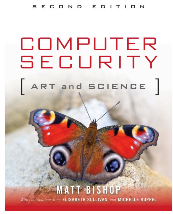
$$(\sigma_a, \sigma_b) \equiv_C (\sigma_c, \sigma_d) \text{ iff } \sigma_a \equiv \sigma_c \text{ and } \sigma_b \equiv \sigma_d$$

- Corresponds to equivalence relation in property 2 for component system



Theorem

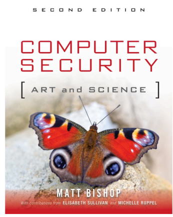
The system resulting from the composition of two restrictive systems is itself restrictive



Side Channels

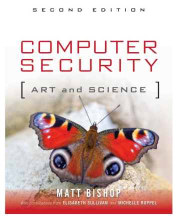
A *side channel* is set of characteristics of a system, from which adversary can deduce confidential information about system or a competition

- Consider information to be derived as HIGH
- Consider information obtained from set of characteristics as LOW
- Attack is to deduce HIGH values from LOW values only
- Implication: attack works on systems not deducibly secure



Types of Side Channel Attacks

- *Passive*: Only observe system; deduce results from observations
- *Active*: Disrupt system in some way, causing it to react; deduce results from measurements of disruption



Example: Passive Attack

- Fast modular exponentiation:

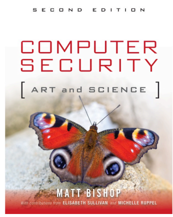
```
x := 1; atmp := a;
for i := 0 to k-1 do begin
   if z_i = 1 then
         x := (x * atmp) mod n;
   atmp := (atmp * atmp) mod n;
end;
result := x;
```

- If bit is 1, there are 2 multiplications; if it is 0, only one
- Extra multiplication takes time
- Can determine bits of the confidential exponent by measuring computation time

Example: Active Attack

Background

- Derive information from characteristics of memory accesses in chip
- Intel x86 caches
  - Each core has 2 levels, L1 and KL2
  - Chip itself has third cache (L3 or LLC)
  - These are hierarchical: miss in L1 goes to L2, miss in L2 goes to L3, miss in L3 goes to memory
  - Caches are inclusive (so L3 has copies of data in L2 and L1)
- Processes share pages



Example: Active Attack

Phase 1

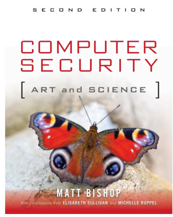
- Flush a set of bytes (called a *line*) from cache to clear it from all 3 caches
  - The disruption

Phase 2

- Wait until victim has chance to access that memory line

Phase 3

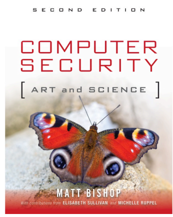
- Reload the line
  - If victim did this already, time is short as data comes from L3 cache
  - Otherwise time is longer as memory fetch is required



Example: Active Attack

What happened

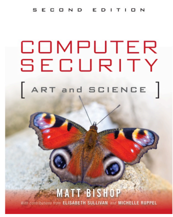
- Used to trace execution of GnuPG on a physical machine
- Derived bits of a 2048 bit private key; max of 190 bits incorrect
- Repeated experiment on virtual machine
- Error rates increased
  - On one system, average error rate increased from 1.41 bits to 26.55 bits
  - On another system, average error rate increased from 25.12 bits to 66.12 bits



Model

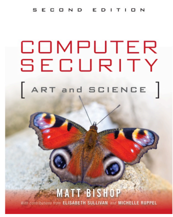
Components

- *Primitive*: instantiation of computation
- *Device*: system doing the computation
- *Physical observable*: output being observed
- *Leakage function*: captures characteristics of side channel and mechanism to monitor the physical observables
- *Implementation function*: instantiation of both device, leakage function
- *Side channel adversary*: algorithm that queries implementation to get outputs from leakage function



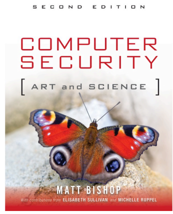
Example

- First one (passive attack) divided leakage function into two parts
  - *Signal* was variations in output due to bit being derived
  - *Noise* was variations due to other factors (imprecisions in measurements, etc.)
- Second one (active attack) had leakage function acting in different ways
  - Physical machine: one chip used more advanced optimizations, thus more noise
  - Virtual machine: more variations due to extra computations running the virtual machines, hence more noise



Example: Electromagnetic Radiation

- CRT video display produces radiation that can be measured
- Using various equipment and a black and white TV, van Eck could reconstruct the images
  - Reconstructed pictures on video display units in buildings
- E-voting system with audio activated (as it would be for visually impaired voters) produced interference with sound from a nearby transistor radio
  - Testers believed changes in the sound due to the interference could be used to determine how voter was vioting



Key Points

- Composing secure policies does not always produce a secure policy
  - The policies must be restrictive
- Noninterference policies prevent HIGH inputs from affecting LOW outputs
  - Prevents “writes down” in broadest sense
- Nondeducibility policies prevent the inference of HIGH inputs from LOW outputs
  - Prevents “reads up” in broadest sense
- Side channel attacks exploit deducability