Today & the next several classes

- Security policies
  - how to classify & reason about them in general
- Security policies that can be enforced at run time
  - security automata
  - stack inspection
  - concurrency

Example Security Policies

- Access control
  - certain principals can/cannot perform certain operations on objects
- Information flow
  - certain principals can/cannot infer information about certain objects by observing system behavior
- Availability
  - principals cannot deny others the use of certain resources

Q: Are there any differences between these policies? Ease/requirements on enforcement? Mechanisms for enforcement of general classes of policies?

Modeling Systems

- want to talk about policies independent of any particular language

- "target" = the system we want to be secure (S)

- execution = a finite / infinite sequence of events / states

  eg:

  open(f); read(f); read(f); write(f); send(z); ....

- Ψ = set of all possible executions

- $\Sigma_S \subseteq \Psi$ defines the executions of the system S

Definition of Security Policy

- A security policy is a predicate on sets of executions.

- A target S satisfies a policy P iff $P(\Sigma_S) = \text{true}$

Examples

- Access Control for files

$\Psi = (read(f) \mid write(f))^*$

$P_M(\Sigma_s) = \forall \sigma \in \Sigma_s .$

if $read(f) \in \sigma$ then

$r \in M(f)$

if $write(f) \in \sigma$ then

$w \in M(f)$

Information Flow

- φ : input(password) ; a^* ; broadcast(s)

- $P(\Sigma_s)$:

  if input(p) ; a^* ; broadcast(s) $\in \Sigma_s$

  then input(p') ; a^* ; broadcast(s) $\in \Sigma_s$

  for arbitrarily many other p'

  ie: the string s that we

  broadcast is independent of p.

Availability

$\psi = \text{access}(p, r)^*$

principal — resource

$P(\Sigma_s) = \forall \sigma \in \Sigma_s .$

$\quad$ if $\text{access}(p, r) = \sigma[i]$

$\quad$ then $\exists j \overset{?}{>} i .\ \text{access}(q, r) = \sigma[j]$

- Eventually another principal q gets access to resource r

Refining Policies

- A property is security policy defined to hold on all individual executions of a system.

$$P(\Sigma) = \forall \sigma \in \Sigma \,.\, \hat{P}(\sigma)$$

- Access control is a property
- Available is a property
- Information flow is not

Safety Properties

- "Nothing bad happens"
- Formally, $\hat{P}$ is a safety property iff

$\forall \sigma \in \Sigma .$

$\neg \hat{P}(\sigma)$ implies

$\exists i . \forall \tau \in \Psi .$

$\neg \hat{P}(\underbrace{\sigma[..i]}_{\text{the first } i \text{ steps of } \sigma};\tau)$

- Once the bad thing has happened, you are done! No way to reverse it, make things "good"
- Lamport, 1985

The Class EM (Fred Schneider)

- Security policies may be enforced by monitoring execution of a system
- Before each event, monitor checks to determine if event is allowed. If so, terminates execution.
- What class of policies can be enforced?
  - Clearly properties
  - $\forall \sigma \in \Sigma. \ldots$
  - A subset of the safety properties:

$$\forall \sigma \in \Sigma.$$

$$\sigma \notin \Gamma \Rightarrow (\exists i. \forall \tau \in \Psi. \sigma[..i]\tau \notin \Gamma)$$

monitor rejects at i

Example EM Properties

EM property:

Access Control

$\forall \sigma \in \Sigma .$

$$\sigma \notin \Gamma \Rightarrow \sigma = \overbrace{a^* ; access(p,r)}^{i} ; b^*$$

$$\Rightarrow \exists i .\ \forall \tau .\ \sigma[i]\tau \notin \Gamma$$

$$\Rightarrow \forall \tau .\ \exists i .\ \sigma[i]\tau \notin \Gamma$$

Non EM

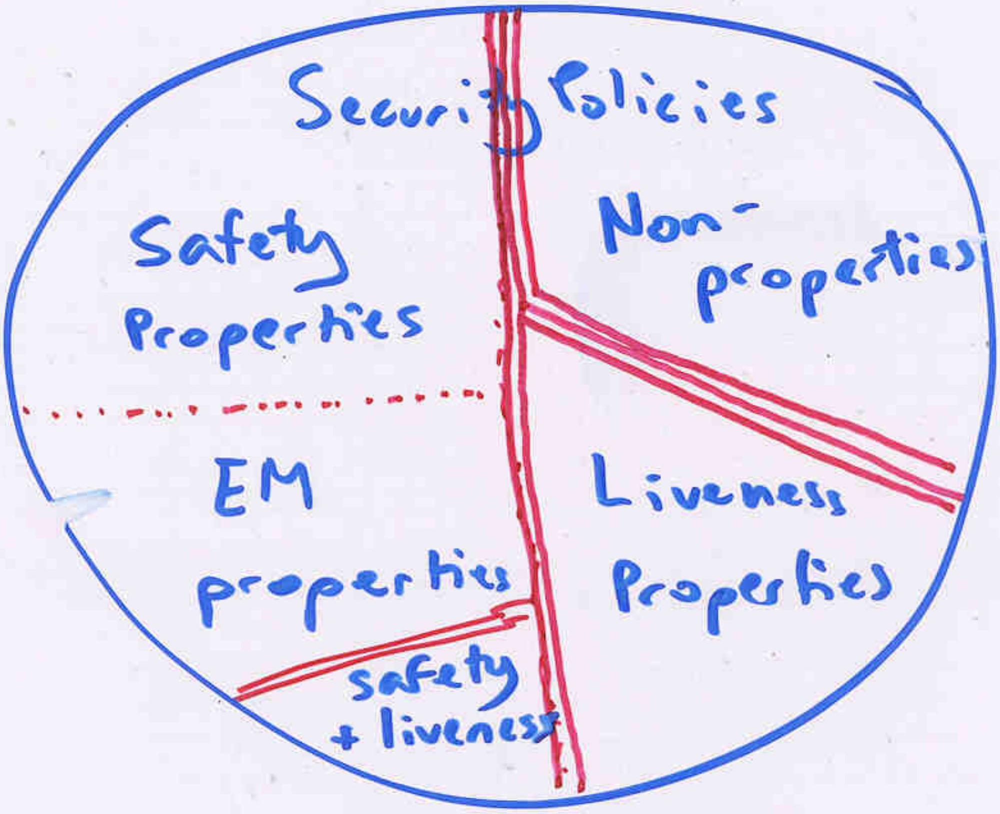
→ Bounded Availability

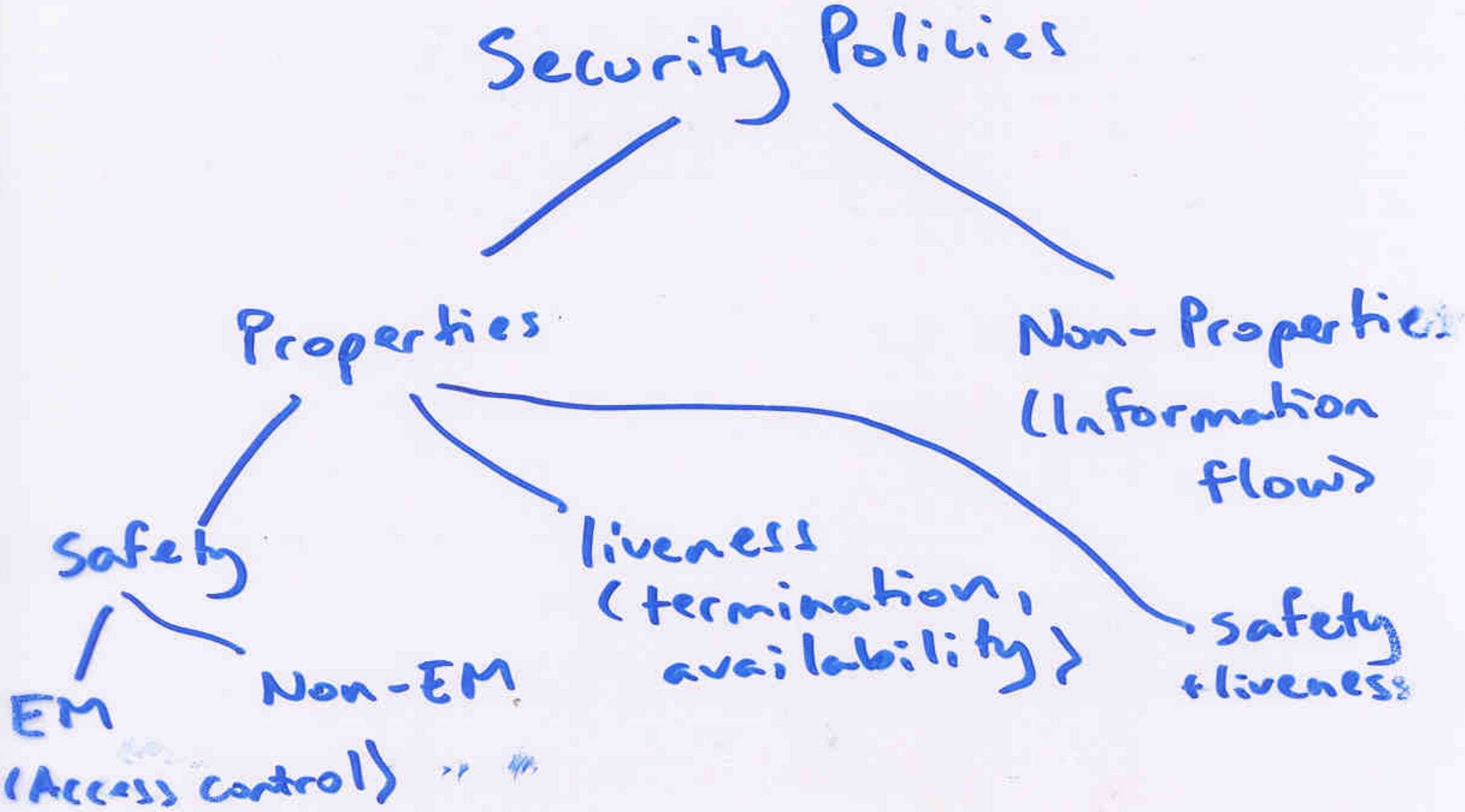
$$\forall \sigma \in \Sigma .\ \sigma \notin \Gamma \Rightarrow \sigma = a^* ; acquire(l) ; a^j ; release(l) ; b^*$$

and $j < 17$ ← Finite bound

- But this is a safety property!

Summary So Far





An Enforcement Mechanism for EM properties

- We require a recognizer that processes the sequence of system events and recognizes policies

- A Security Automaton is such a recognizer

- S.A. definition:

  Q = set of automaton states

  q_0 = initial state

  I : input symbols

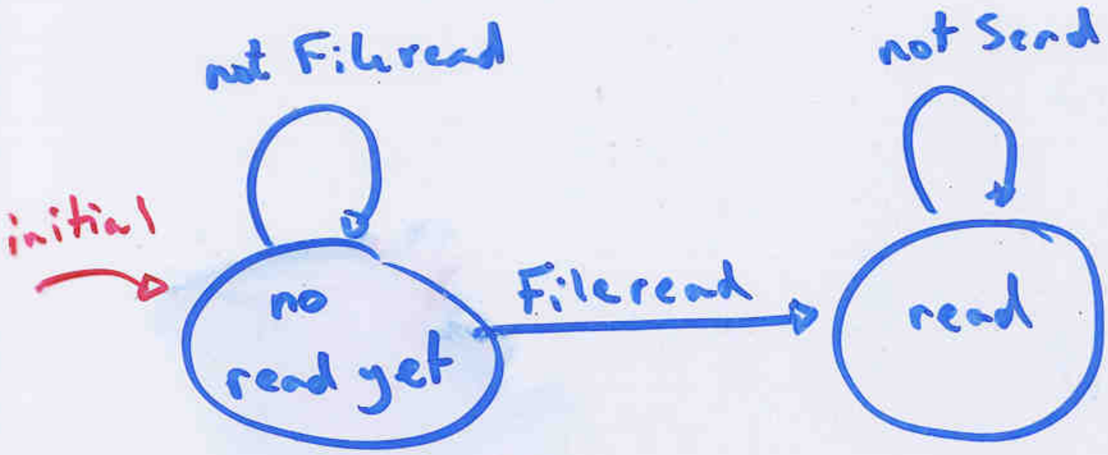
  δ : transition function/relation

  $Q \times I \to 2^Q$

when $\delta(q, i) = \emptyset$

we say the automaton rejects the input i in state q and the policy is violated.

Example Automaton



Policy: No send on network after file read.

$Q = \{$no-read-yet, read$\}$

$q_0 =$ no-read-yet

$I = \{$Fileread, Send, ...$\}$

$\delta =$

A Simple Language for Specifying Automata

- pgm ::=

  let state =

  $s1 = v_1$ $\Big\}$ state variables

  $\vdots$

  $s_n = v_n$

  in

  $C_1 \cdots C_n$ $\}$ Commands

- $C ::= i : B \to S$

- $B ::= p(s_1, \cdots, s_n, i)$

  $B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B \mid$

  $B_1 \Rightarrow B_2 \mid$

- $S ::= s_1 := f(s_1 \cdots s_n) \mid$

  if B then S_1 else $S_2 \mid$

  $S_1 ; S_2 \mid$

  skip

$V ::= i \mid \{\circ\} \mid$

Example Spec

let

  state = 0

in

i: (not Fileread(i)) $\wedge$ ($Equals_0$(state))

  $\rightarrow$ skip ,

i: Fileread(i) $\wedge$ $Equals_0$ (state)

  $\rightarrow$ state := 1 ,

i: not send(i) $\wedge$ $Equals_1$ (state)

  $\rightarrow$ skip

Another Example

- Access Control
- $p \in$ PRINCIPALS

  $o \in$ OBJECTS

  $r \in$ RIGHTS
- A is the access control matrix

  $\langle p, o, r \rangle \in A$ iff p has right r to object o

- Predicates

  oper (p, o, r, i)
  - p invokes operation i that requires a right r on o

  addright (p, p', r', o', i)
  rmvright (p, p', r', o', i)
  addP (p, p', i) addO (p, o', i)
  rmvP (p, p', i) rmvO (p, o', i)

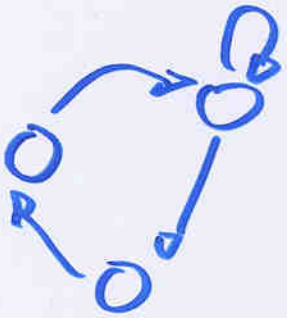
```
let
    P = ∅        % set of p ∈ PRINCIPALS
    O = ∅        % set of o ∈ OBJECTS
    A = ∅        % set of   <s:PRINCIPALS
                 %           o:OBJECTS
                 %           r:RIGHTS>

in

  i:oper(p,o,r,i) ∧ <p,o,r> ∈ A
       → skip
  i:AddRight(p,p',r',o',i) ∧ <p,o',cntrl> ∈ A
       → A := A ∪ {<p',o',r'>}
  i:RmvRight(p,p',r',o',i) ∧ <p,o',cntrl> ∈ A
       → A := A − {<p',o',r'>}
  i:AddP(p,p',i)
       → P := P ∪ {p'} ; O := O ∪ {p'} ;
         A := A ∪ {<p,p',cntrl>}
       ⋮
```

Composing Automata

- When specifying the security policy for a large system we might consider a single aspect at a time



A_1: Access Control

A_2: Resource Bounds

- The complete policy is the conjunction of automata

$A_1 \wedge A_2$

- states : $Q_1 \times Q_2$
- transitions : $\delta(q_1, q_2)(i) = \delta_1(q_1)(i) \times \delta_2(q_2)(i)$
- reject if either transition is undefined

Beyond EM

- Many more policies can be enforced if the automaton is allowed to modify the event stream

- eg:
  - Bounded Availability

  Events: Acquire(r), Release(r)

  $P(\Sigma_S) = \forall \sigma \in \Sigma_S .$

  if $\sigma[i] =$ Acquire(r)

  then $\sigma[i+k] =$ Release(r)

  where $k \leq j$

- Enforcement:
  - Monitor input stream. After j-1 steps following Acquire(r), insert Release(r).

Transactional Policies

- in a database system, we might require certain consistency conditions

- eg

Personal Data Table

Name, Phone, Address, Social Security

Visa Status

Social Security Visa Expiry

Payroll

Name Social Security $ TAX

- Conditions:
  - Every Name-SSN pair in personal data table must have a visa status entry
  - Every Name-SSN pair in Payroll must have Name-SSN in personal data

- Operations

  Add Personal (n, ph, addr, ssn)

  Rmv Personal (n)

  Add V (ssn, visa, ex)

  Rmv V (ssn)

  Add Pay (n, ssn, money, tax)

  Rmv Pay (n)

  Time ()

- if we Add Personal (n, ph, addr, ssn) then we must Add V (ssn, visa, ex) within 3 days. otherwise, we must Rmv Personal(n) on the 4th day

- more generally, we log the database transactions and if our consistency conditions are not met, we roll-back to a consistent state.
- requires invertible operations.

A new Enforcement Mechanism

- Model:
  There are 2 ways we can affect system behavior

  1. Terminate
  2. Insert symbols/events

- Automata now have 2 sorts of states

  Q = accepting states in which we may halt execution

  R = neutral states in which we may not halt execution but for which there is a transition to an accepting state

- Automata Definition

$Q \cup R = T$: total set of states

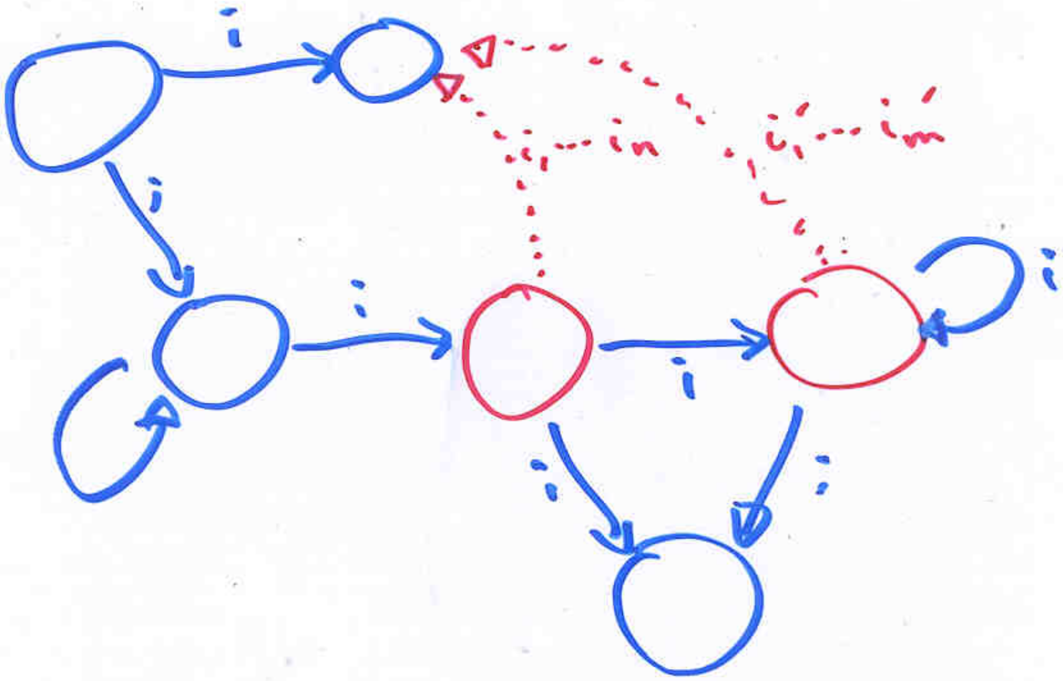
$t_0 \in T$ = initial state

$\underline{I}$: inputs as before

$\delta : T \times i \to 2^T$: normal transitions

$\sigma : R \to \vec{i} \times Q$: exception transitions

Extended Automata



- Blue states as before
  - we may safely halt execution here

- Red states
  - we may not halt
  - but there exists a transition to a safe blue state

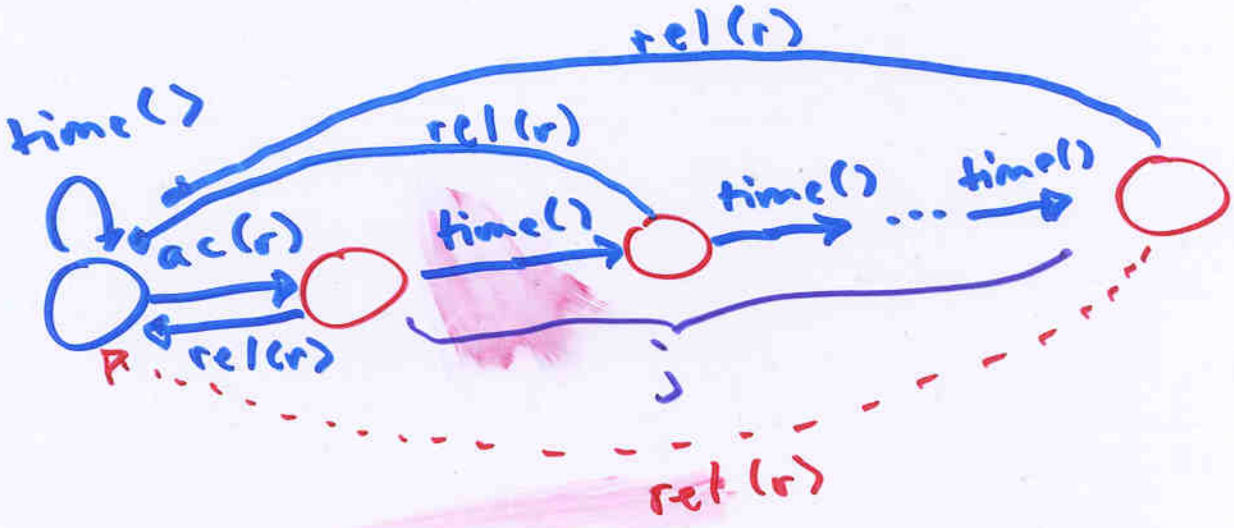
Bounded Availability

Operations

ac(r) - acquire resource r

rel(r) - release resource r

time() - 1 day passes



policy: After j days, resource r must be released

- Extended policies are safety policies not in EM

- recall: P is a safety property iff

$\forall \sigma \in \Sigma_S .$

$\text{not } \hat{P}(\sigma) \Rightarrow \exists i . \forall \tau \in \Psi . \text{ not } \hat{P}(\sigma[..i]\tau)$

Next time

- Practical Implementations
- SASI (Erlingsson & Schneider)
- Naccio (Evans & Twyman)
- Database policies
  - PRISM (Payette et al.)