Today & the next several classes

- Security policies
  - how to classify & reason about them in general
- Security policies that can be enforced at runtime
  - 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
- execution = a finite/infinite sequence of events/states
  eg:
  open(f); read(f); read(f); while(f);
  send(z); ...
- \( \Psi \) = set of all possible executions

\( \Sigma \subseteq \Psi \) defines the executions of the system \( \Sigma \)
Definition of Security Policy

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

- A target $S$ satisfies a policy $P$ iff $P(S) = true$
Examples

- Access Control for files
  \[ \psi = (\text{read}(f) \lor \text{write}(f))^\star \]

\[ p_\omega(\Sigma_\omega) = \forall \theta \in \Sigma_\omega.
\]

  if \( \text{read}(f) \in \theta \) then
  \[ r \in M(f) \]

  if \( \text{write}(f) \in \theta \) then
  \[ w \in M(f) \]
Information Flow

- \[ \psi = \text{input (password)} \ast \text{; broadcast}(s) \]

- \[ \mathcal{P}(E) = \]

  \[
  \text{if } \text{input}(p \ast) \ast \text{; broadcast}(s) \in \mathcal{E} \]
  \[
  \text{then } \text{input}(p') \ast \text{; broadcast}(s) \in \mathcal{E} \]

for arbitrarily many other \( p' \)

ie: the string \( s \) that we broadcast is independent of \( p \).
Availability

\[ y = \text{access}(p, r) \ast \]
\[ \sim \]
principal \quad \text{resource}

\[ P(\Sigma_s) = \forall \sigma \in \Sigma_s . \]
\[ \text{if access}(p, r) = \sigma[i] \]
\[ \text{then } \exists j^2 : \text{access}(q, r) = \sigma[j] \]

- 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

\[ \forall \epsilon \exists \delta. \quad \forall \epsilon' \in \mathcal{E}. \quad \exists \epsilon'' \in \mathcal{E}. \quad \hat{P}(\epsilon'') \implies \hat{P}(\epsilon) \]

\[ \text{i.e. } \forall \epsilon \exists \delta \]

\[ \forall \epsilon' \in \mathcal{E}. \quad \exists \epsilon'' \in \mathcal{E}. \quad \hat{P}(\epsilon''[\ldots \epsilon' \ldots]) \]

- 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
  - \( \forall \alpha \in \Sigma \, \ldots \)
  - A subset of the safety properties:
    \( \forall \alpha \)
    \[ 6 \notin \mathbb{N} \Rightarrow (3 \cdot 4 \cdot \alpha + 6 \cdot \cdot \cdot \cdot \cdot \cdot \cdot \) \]

monitor rejects at ?
Example EM Properties

EM property:

Access Control

\( \forall a \in \Sigma \cdot \)

\( 6 \notin \Gamma \Rightarrow 6 \ast a^* ; \text{access}(p,r) ; 6^* \Rightarrow \exists i \cdot \forall 2 \cdot 6[i]z \notin \Gamma \Rightarrow \forall 2 \cdot \exists i \cdot 6[i]z \notin \Gamma \)

Non-EM

\( -Bounded\ \text{Availability} \)

\( \forall a \in \Sigma \cdot 6 \notin \Gamma \Rightarrow 6 \ast a^* ; \text{acquire}(2) ; a^j ; \text{release}(2) ; 6^* \) and \( j < 17 \) - Finite bound

-But this is a safety property!
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

  \( S = \) transition function/relation

  \[ Q \times I \rightarrow 2^Q \]

  when \( S(q, i) = \emptyset \)

  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.

\[ Q = \{ \text{no-read-yet}, \text{read} \} \]
\[ q_0 = \text{no-read-yet} \]
\[ I = \{ \text{File read}, \text{Send} \} \]
\[ J = \]
A Simple Language for Specifying Automata

- $\text{pgm}::=
  \text{let state =}
  \begin{align*}
  s_1 &= v_1 \\
  \vdots \\
  s_n &= v_n \\
  \text{in}
  C_1, \ldots, C_n
  \end{align*}

\begin{align*}
\text{state variables} \\
\text{Commands}
\end{align*}

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

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

\begin{align*}
B, &\ A B_2 \ 1 \ B, \ v B_2 \ 1 \ \ 2 B \ 1 \\
B, \ \Rightarrow &\ 0 B_2 \ 1
\end{align*}

- $S ::= s, := f(s_1, \ldots, s_n) \ 1$

\begin{align*}
\text{if } B \text{ then } s, &\text{ else } s_2, 1 \\
S_1, &\ j S_2, 1 \\
\text{skip}
\end{align*}

- $V ::= i \mid \exists v 3 \ 1$
Example Spec

let
    state = 0
in
i: (not FileRead(i)) \land (Equals_0(state))
   \rightarrow skip;

i: FileRead(i) \land Equals_0(state)
   \rightarrow state := 1;

i: not Send(i) \land Equals_1(state)
   \rightarrow skip
Another Example

- Access Control
  - \( p \in PRINCIPALS \)
  - \( o \in OBJECTS \)
  - \( r \in RIGHTS \)

- \( A \) is the access control matrix
  \( (p, o, r) \in A \) iff \( p \) has right \( r \) to object \( o \)

- Predicates
  - \( oper \ (p, o, r, i) \):
    - \( p \) invokes operation \( i \) that requires a right \( r \) on \( o \)
  - \( addright \ (p, o, r, i) \)
  - \( rmvright \ (p, o, r, i) \)
  - \( addP \ (p, p', i) \)
  - \( addO \ (p, o, i) \)
  - \( rmvP \ (p, p', i) \)
  - \( rmvO \ (p, o, i) \)
\[
\begin{align*}
\text{let } & \quad p = \emptyset \quad \% \text{ set of PRINCIPALS} \\
\text{let } & \quad o = \emptyset \quad \% \text{ set of OBJECTS} \\
\text{let } & \quad A = \emptyset \quad \% \text{ set of } \langle s: \text{PRINCIPALS} \\
& \quad o: \text{OBJECTS} \\
& \quad r: \text{RIGHTS} \rangle \\
\text{in} \\
\langle \text{oper} (p, o, r, i) \land \langle p, o, r \rangle \in A \\
\rightarrow \text{skip} \\
\langle \text{AddRight} (p, p', r', o', i) \land \langle p, o, \text{cntrl} \rangle \in A \rightarrow A := A \cup \{ \langle p', o', r' \rangle \} \\
\langle \text{RmvRight} (p, p', r', o', i) \land \langle p, o, \text{cntrl} \rangle \in A \rightarrow A := A \setminus \{ \langle p', o', r' \rangle \} \rangle \\
\langle \text{AddP} (p, p', i) \rightarrow p := p \cup \{ p' \}; o := o \cup \{ p' \} \\
A := A \cup \{ \langle p, p', \text{cntrl} \rangle \} \rangle
\end{align*}
\]
Composing Automata

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

\[ A_1 : \text{Access Control} \]
\[ A_2 : \text{Resource Bounds} \]

- The complete policy is the conjunction of automata

\[ A_1 \land A_2 \]
- states: \( Q_1 \times Q_2 \)
- transitions: \( S(q_1, q_2)(i) = S(q_1)(i) \times S(q_2)(i) \)
- 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: \text{Acquire}(r), \text{Release}(r)

  \[ P(E) = \forall 6 \in E \]

  if \[ 6[i:j] = \text{Acquire}(r) \]
  then \[ 6[i+\ell:j] = \text{Release}(r) \]
  where \( \ell \leq j \)

- Enforcement:
  - Monitor input stream. After \( j-1 \) steps following \text{Acquire}(r), insert \text{Release}(r). \]
**Transactional Policies**

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

  - e.g.

<table>
<thead>
<tr>
<th><strong>Personal Data Table</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name, Phone, Address, Social Security</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Visa Status</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Social Security, Visa, Expiry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Payroll</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name, Social Security</td>
</tr>
</tbody>
</table>

- 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, fx)
  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
to \( E \in T \) : initial state

\( I \) : inputs as before

\( S : T \times i \rightarrow 2^T \) : normal transitions

\( G : R \rightarrow i \times Q \) : exception transitions
- 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
- \text{ac}(r) \quad \text{acquire resource}
- \text{rel}(r) \quad \text{release resource}
- \text{time}() \quad \text{1 day passes}

Policy: After \emph{j} days, resource \emph{r} must be released
- Extended policies are not in EM
- safety policies not in EM
- recall: P is a safety property iff

∀s ∈ S:
not ∃i. ∀j ≥ 4. not P(σ[i...i])
Next time

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