More Enforceable Security Policies
by Lujo Bauer, Jarred Ligatti and David Walker

Presented by Khoo Yit Phang
April 21, 2008
To Build Secure Systems...

1. What sort of security policies can and should we demand of our system?

2. What mechanism should we implement to enforce these policies?
Execution Monitoring (EM)

- EM is a runtime security automaton.
- EM can be shown to enforce safety properties (Schneider 2000; last week’s paper)
- EM is limited because it can only terminate unsafe programs.
More Enforceable Security Policies

Extend EM by introducing automatons that modify a program sequence:

1. Insertion automaton
2. Suppression automaton
3. Edit automaton = Insertion + Suppression
Review: Policies and Properties

Security policy
- a set of executions $\Sigma$ satisfies policy $P$ iff $P(\Sigma)$
- defined over all executions
- e.g. information flow

Security property
- $P$ is a property iff $P(\Sigma) = \forall \sigma \in \Sigma. \hat{P}(\sigma)$
- where $\hat{P}$ is a predicate on uniform systems (finite sequence of program actions)
- defined over individual executions
- e.g. access control, availability, bounded availability
Properties are conjunctions of safety and liveness:

- **Safety** – “nothing bad happens” (e.g. access control)
  \[\neg \hat{P} (\sigma) \Rightarrow \forall \sigma' \in \Sigma. (\sigma < \sigma' \Rightarrow \neg \hat{P} (\sigma'))\]

- **Liveness** – “something good must happen” (e.g. availability)
  \[\forall \sigma \in \Sigma. \exists \sigma' \in \Sigma. (\sigma < \sigma' \land \hat{P} (\sigma'))\]

- **Safety + Liveness** – “something good must happen by x” (e.g. bounded availability)
Precise Enforcement

- An automaton *precisely* enforces \( \hat{P} \) iff \( \forall \sigma \in \Sigma \)

1. If \( \hat{P}(\sigma) \) then \( \forall i. (\sigma, q_0) \xrightarrow{\sigma[i..]} A (\sigma[i + 1..], q') \) and,

2. If \( (\sigma, q_0) \xrightarrow{\cdot} A (\cdot, q') \) then \( \hat{P}(\sigma') \)

1. does not modify an allowed sequence
2. must edit an unallowed sequence to conform to \( \hat{P} \)

- An automaton *conservatively* enforces \( \hat{P} \) if it does not hold condition 1
  - may be disruptive to a correct program
Enforced by security automaton FSA \((Q, q_0, d)\)
- \(Q\): states
- \(q_0\): initial state
- \(d\): transition function

**A-Step**
- step if \(a\) (prefix of \(\tau\)) is an allowed sequence

**A-Stop**
- stop if \(\tau\) has no allowed sequence
Beyond EM: **Insertion**

**Insertion function** $\gamma$

**I-Step, I-Stop**
- like A-Step, A-Stop

**I-Ins**
- insert $\tau$ if not I-Step and $\gamma(a, q) = \tau, q'$

E.g., bounded-availability:
- insert *release* after $n$ uses/end of program

\[
\begin{align*}
\sigma, q &\xrightarrow{a} I (\sigma', q') \quad \text{(I-Step)} \\
\text{if } \sigma = a; \sigma' &\quad \text{and } \delta(a, q) = q' \\
\sigma, q &\xrightarrow{\tau} I (\sigma, q') \quad \text{(I-Ins)} \\
\text{if } \sigma = a; \sigma' &\quad \text{and } \gamma(a, q) = \tau, q' \\
\sigma, q &\xrightarrow{.} I (., q) \quad \text{(I-Stop)} \\
\text{otherwise}
\end{align*}
\]
Beyond EM: Suppression

Suppression function $\omega$

**S-StepA**
- if $\omega(a,q) = +$ like A-Step

**S-Stop**
- like A-Stop

**S-StepS**
- suppress program action if $\omega(a,q) = -$ 

E.g. suppress *use* after *n* uses, leave *release* alone.

For any suppression automaton, can construct an equivalent insertion automaton

\[
(\sigma, q) \xrightarrow{a} S (\sigma', q') \quad (S-\text{STEP A})
\]

if $\sigma = a; \sigma'$
and $\delta(a, q) = q'$
and $\omega(a, q) = +$

\[
(\sigma, q) \xrightarrow{.} S (\sigma', q') \quad (S-\text{STEP S})
\]

if $\sigma = a; \sigma'$
and $\delta(a, q) = q'$
and $\omega(a, q) = -$ 

\[
(\sigma, q) \xrightarrow{.} S (\cdot, q) \quad (S-\text{STOP})
\]

otherwise
Beyond EM: *Edit*

Edit = Insert + Suppress

E-StepA, E-StepS
- like S-StepA, S-StepS

E-Ins
- like I-Ins

E-Stop
- like A-Stop

\[(\sigma, q) \xrightarrow{a}_E (\sigma', q') \quad \] (E-StepA)

if \( \sigma = a; \sigma' \)
and \( \delta(a, q) = q' \)
and \( \omega(a, q) = + \)

\[(\sigma, q) \xrightarrow{\tau}_E (\sigma, q') \quad \] (E-StepS)

if \( \sigma = a; \sigma' \)
and \( \delta(a, q) = q' \)
and \( \omega(a, q) = - \)

\[(\sigma, q) \xrightarrow{\gamma}_E (\cdot, q) \quad \] (E-Ins)

if \( \sigma = a; \sigma' \)
and \( \gamma(a, q) = \tau, q' \)

otherwise
For all 3 automata,

If $S$ is a uniform system, and automata $A$ precisely enforce $^\wedge P$ on $S$, then $^\wedge P$ obeys safety
Limitations

- All automata limited by their ability to insert/suppress, e.g.:
  - cannot insert encrypted actions
  - cannot suppress input
Example: Transactions

Enforce ACID properties

*Atomicity*: take(n); pay(n) completes together, or never; suppress initial take(n), and re-insert before pay(n)

*Consistency*: take(n); pay(n) has the same value for n

*Durability*: transactions cannot be reverted after complete (doesn’t durability mean that a new transaction cannot munge an old one?)

*Isolation*: not in this example
Other issues

• How to compose edit automata?
  • simple with EM – programs just terminate
• Edit automata modifies programs
  • Safety properties enforced, but program may become “incorrect”
• What does it mean to effectively enforce a property?
• Can suppression automata enforce properties not by insertion?