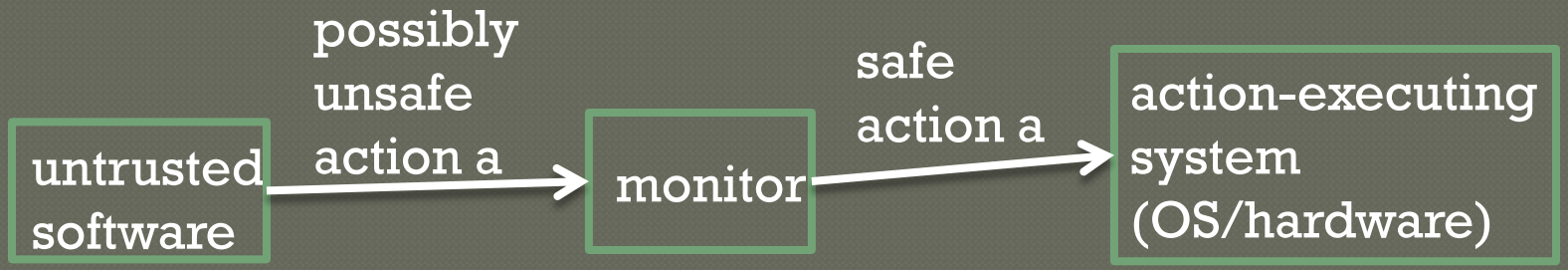
Modeling Enforcement Mechanisms with Security Automata

Jay Ligatti
University of South Florida

Runtime Enforcement Mechanisms, for Software

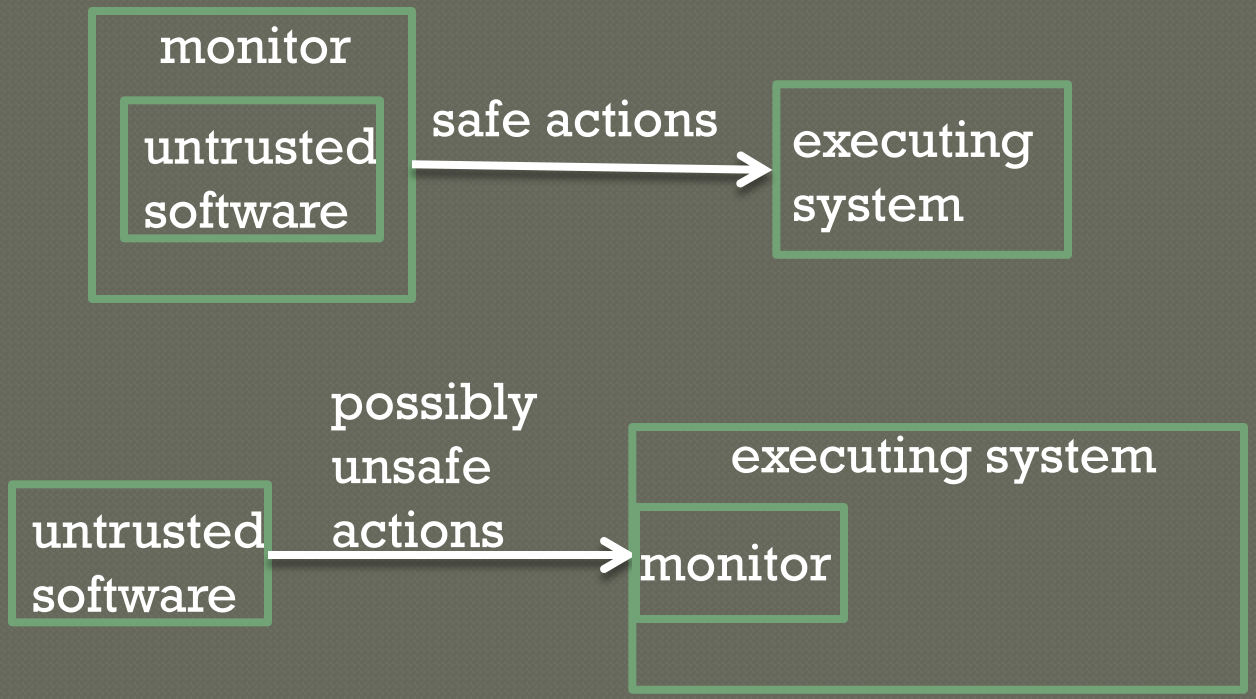
- Interpose on the actions of some untrusted software
- Have authority to decide whether and how to allow those actions to be executed
- Are called runtime/security/program *monitors*



a=open(file,"r") | shutdown() | login(sn,pw) | connect(addr,port) |…

Runtime Enforcement Mechansisms

- Monitoring code can be inserted into the untrusted software or the executing system



Runtime Enforcement Mechanisms

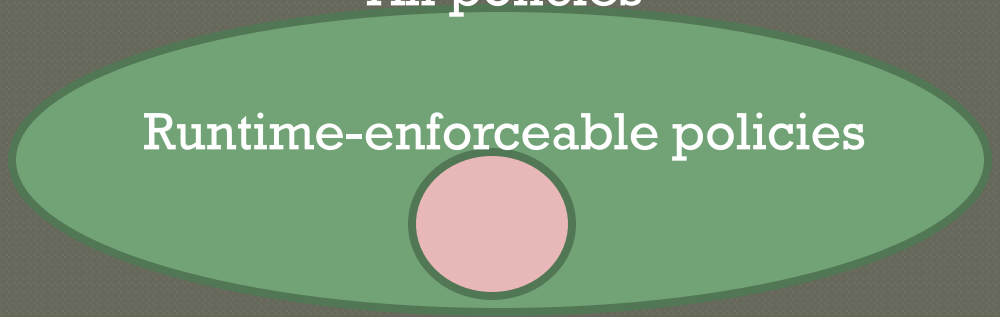
- In all cases monitor inputs possibly unsafe actions from the untrusted software and outputs safe actions to be executed

Runtime Enforcement Mechanisms

- Ubiquitous
  - Operating systems (e.g., file access control)
  - Virtual machines (e.g., stack inspection)
  - Web browsers (e.g., javascript sandboxing)
  - Intrusion-detection systems
  - Firewalls
  - Auditing tools
  - Spam filters
  - Etc.
- Most of what are usually considered "computer security" mechanisms can be thought of as runtime monitors

Research Questions

- How do monitors operate to enforce policies?
  - Which policies can runtime mechanisms enforce?
  - Which policies should we never even try to enforce at runtime?

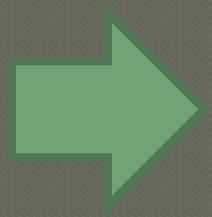


Research Questions

- How do monitors operate to enforce policies?
  - Which policies get enforced when we combine runtime mechanisms?

mechanism M enforces policy P

mechanism M' enforces policy P'



M ^ M' enforces? P ^ P' ?

What if P requires the first action executed to be fopen(f), but P' requires the first action executed to be fopen(f')?

Research Questions

- How do monitors operate to enforce policies?
  - How efficiently does a mechanism enforce a policy?
  - What are the lower bounds on resources required to enforce policies of interest?

What does it mean for a mechanism to be efficient?

- Low space usage
  (SHA of Fong, BHA of Talhi, Tawbi, and Debbabi)
- Low time usage
  ?

Research Questions, Summary

- How do monitors operate to enforce policies?
  - Which policies can runtime mechanisms enforce?
  - Which policies get enforced when we combine runtime mechanisms?
  - How efficiently does a mechanism enforce a policy?
  - What are the lower bounds on resources required to enforce policies of interest?

This Talk

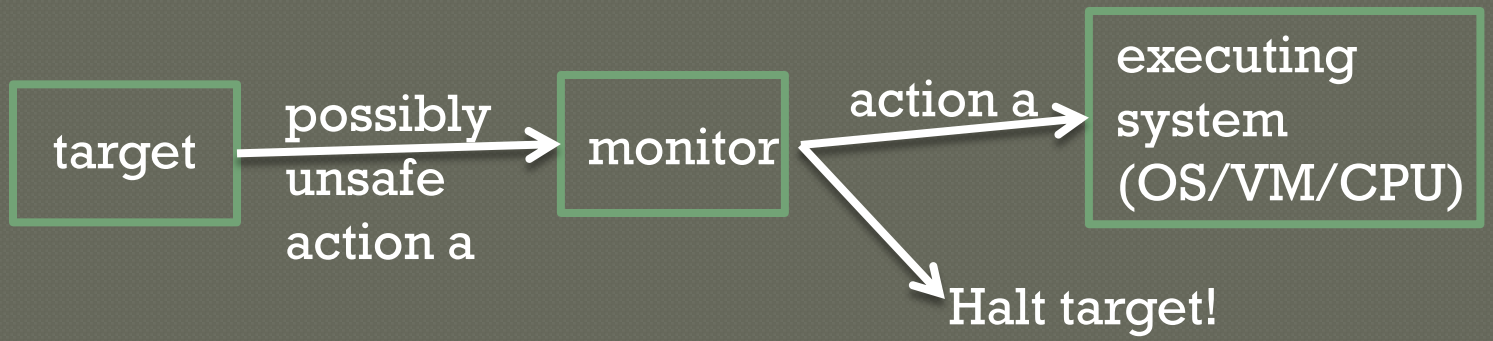
- How do monitors operate to enforce policies?
  - Which policies can runtime mechanisms enforce?
  - Which policies get enforced when we combine runtime mechanisms?
  - How efficiently does a mechanism enforce a policy?
  - What are the lower bounds on resources required to enforce policies of interest?

Outline

- Research questions
  - How do monitors operate to enforce policies?
    - Which policies can runtime mechanisms enforce?
- Related work vs. this work
- The model: systems, executions, monitors, policies, and enforcement
- Analysis of enforceable properties
- Summary and future work

Related Work: Truncation Automata

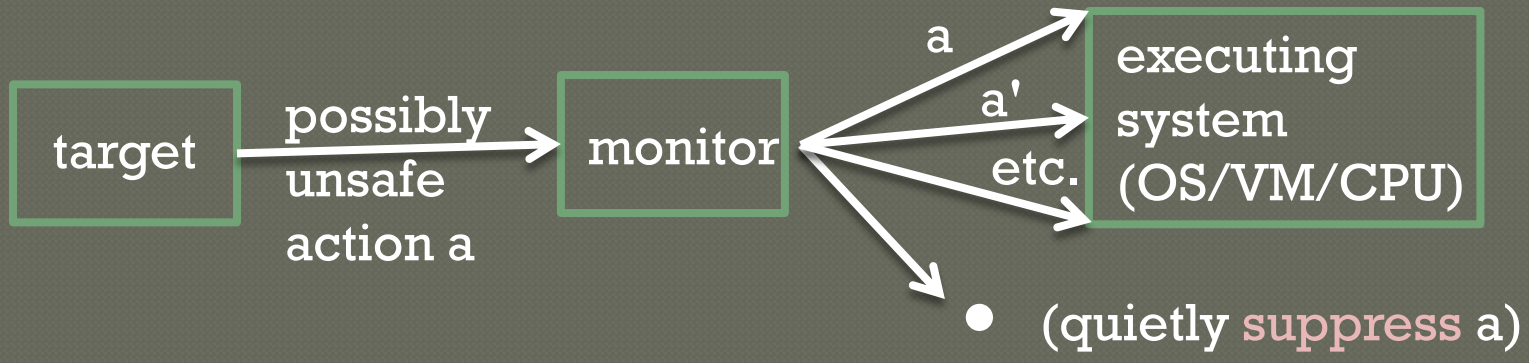
- Most analyses of monitors are based on truncation automata (Schneider, 2000)



- Operation: halt software being monitored (target) immediately before any policy violation
- Limitation: real monitors normally respond to violations with remedial actions

Related Work: Edit Automata

- Powerful model of runtime enforcement



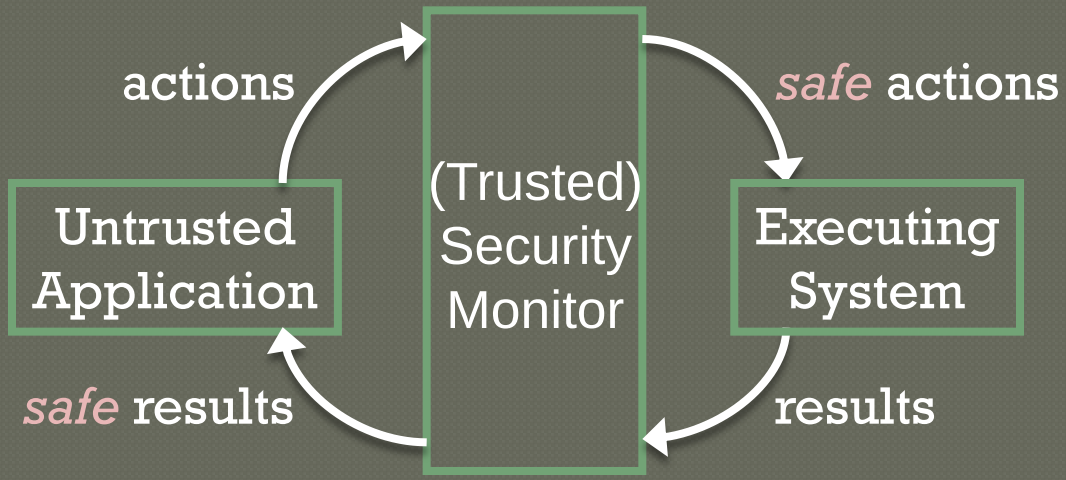
- Operation: actively transform target actions to ensure they satisfy desired policy

Related Work: Edit Automata

- Limitation:
  - All actions are assumed totally asynchronous
    - Monitor can always get next action after suppressing previous actions
    - Target can't care about results of executed actions; there are no results in the model
  - E.g., the echo program "x=input(); output(x);" is outside the edit-automata model

This Work: Mandatory Results Automata (MRAs)

- Conservatively assume all actions are *synchronous* and monitor those actions and their results



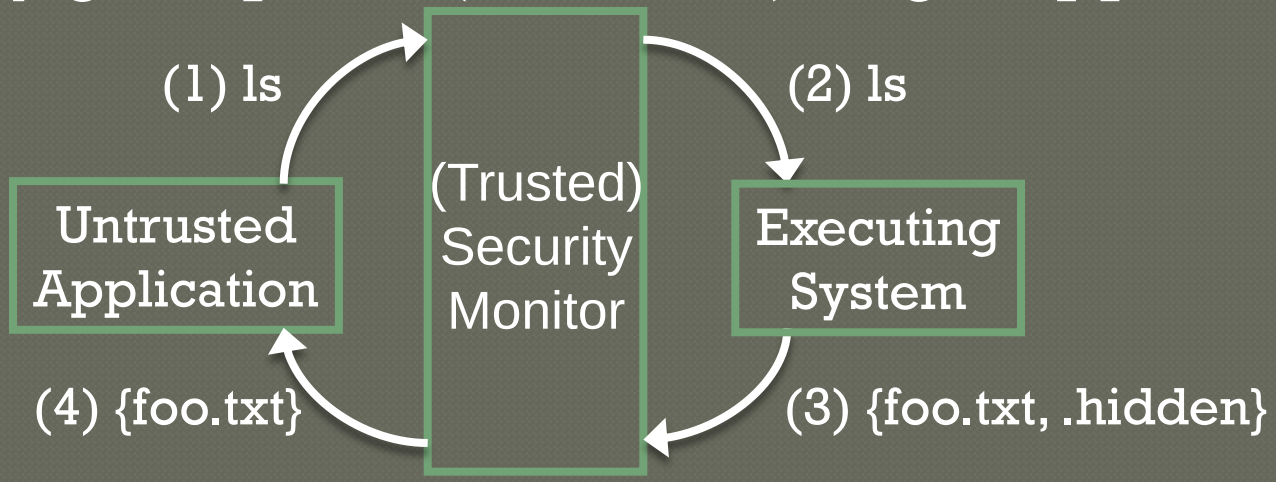
- Operation: actively transform actions and results to ensure they satisfy desired policy

This Work: Mandatory Results Automata (MRAs)

- MRAs are stronger than truncation automata
  - Can accept actions and halt targets but can also *transform* actions and results
- MRAs are weaker than edit automata
  - Asynchronicity lets edit automata "see" arbitrarily far into the future
    - Can postpone deciding how to edit an action until later
    - Arbitrary postponement is normally unrealistic

Other Neat Features of the MRA Model

1. MRAs can enforce result-sanitization policies
   - (trusted) mechanism sanitizes results before they get input to (untrusted) target application



   - Many privacy, information-flow, and access-control policies are result-sanitization

Other Neat Features of the MRA Model

2. Model provides simpler and more expressive definitions of *policies* and *enforcement* than previous work
   - (more on this later)

Outline

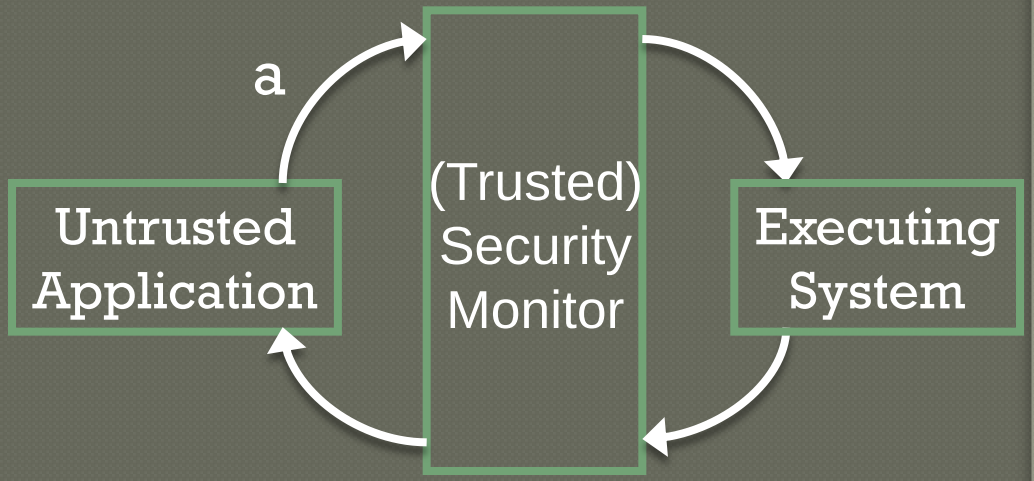
- Research questions
  - How do monitors operate to enforce policies?
    - Which policies can runtime mechanisms enforce?
- Related work vs. this work
- The model: systems, executions, monitors, policies, and enforcement
- Analysis of enforceable properties
- Summary and future work

Systems

- Systems are specified as sets of *events*
  - Let A be a finite or countably infinite set of *actions*
  - Let R (disjoint from A) be a finite or countably infinite set of action *results*
  - Then a *system* is specified as $E = A \cup R$
- Example:
  - A = {popupWindow("Confirm Shutdown"), shutdown()}
  - R = {OK, cancel, null}

Definition of MRA traces/executions

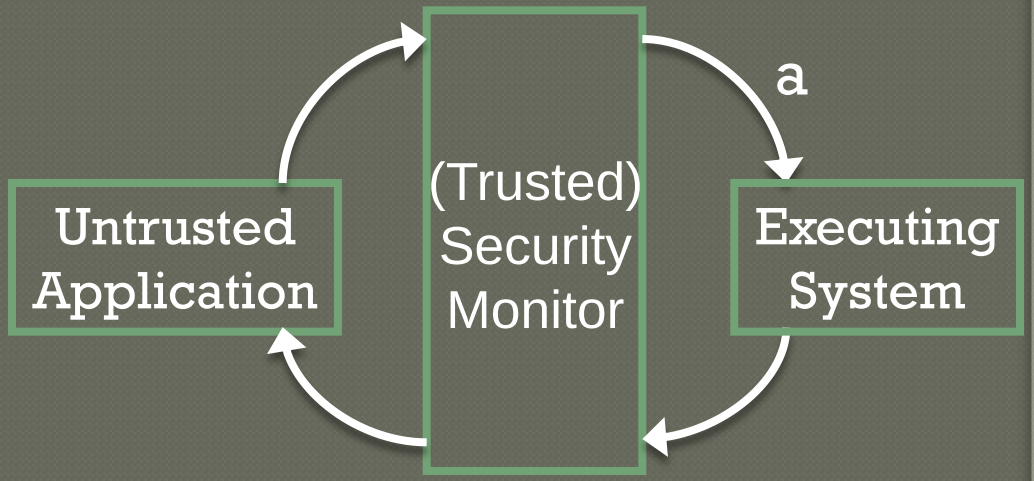
- Execution: finite/infinite sequence of events
- Adopting a monitor-centric view,
  ∃ 4 event possibilities:
  (1) MRA *inputs* action *a* from the target



=> add a_i to the current trace

Definition of MRA traces/executions

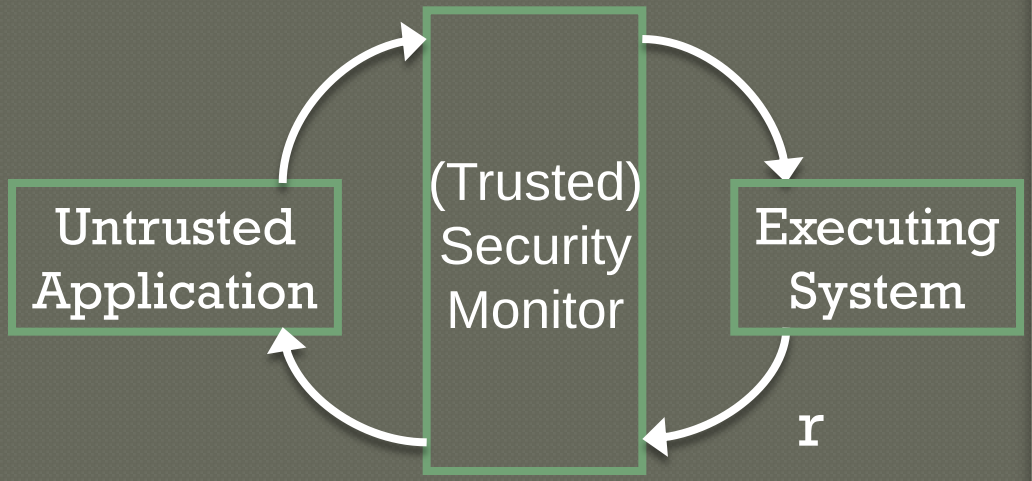
- Execution: finite/infinite sequence of events
- Adopting a monitor-centric view,
  ∃ 4 event possibilities:
  (2) MRA *outputs* action *a* to be executed



=> add a_o to the current trace

Definition of MRA traces/executions

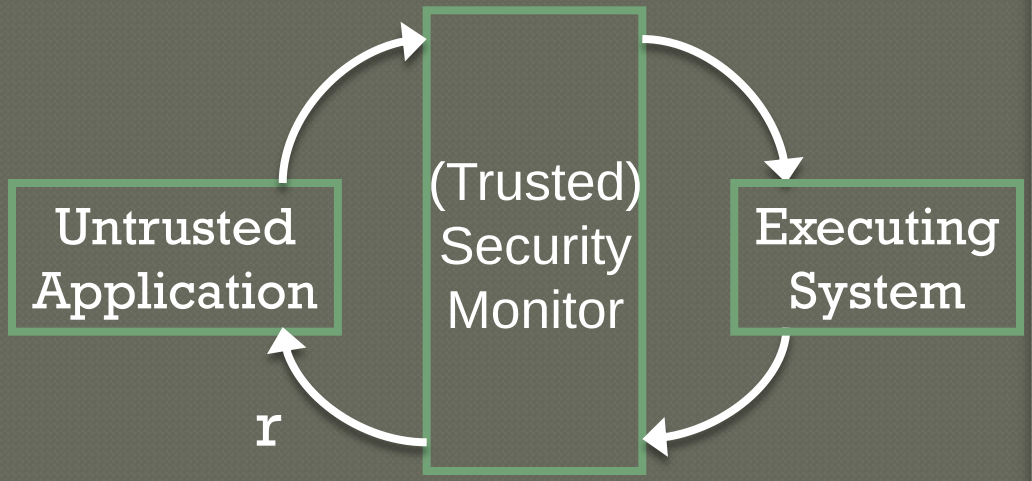
- Execution: finite/infinite sequence of events
- Adopting a monitor-centric view,
  ∃ 4 event possibilities:
  (3) MRA *inputs* result *r* from the system



=> add r_i to the current trace

Definition of MRA traces/executions

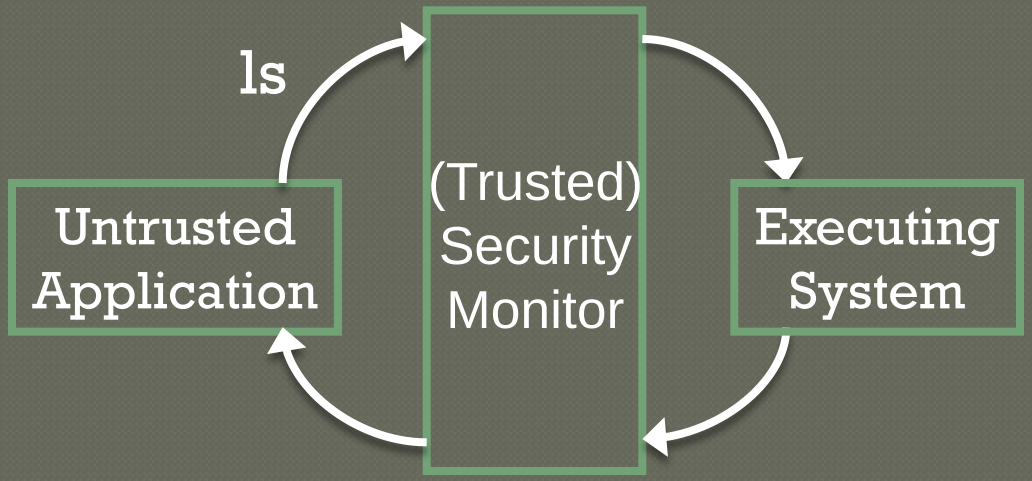
- Execution: finite/infinite sequence of events
- Adopting a monitor-centric view,
  ∃ 4 event possibilities:
  (4) MRA *outputs* result *r* to the target



=> add r_o to the current trace

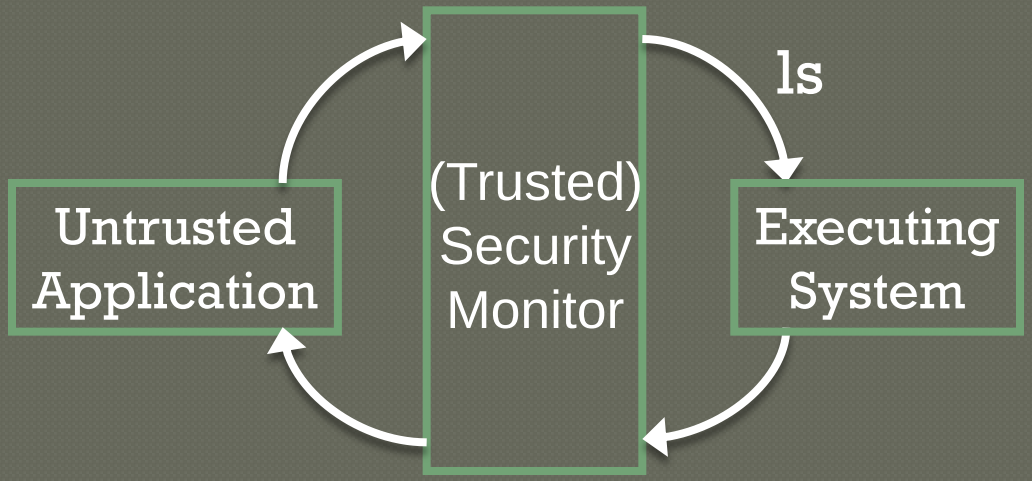
Example Execution

- $ls_i ; ls_o ; \{foo.txt, .hidden\}_i ; \{foo.txt\}_o$



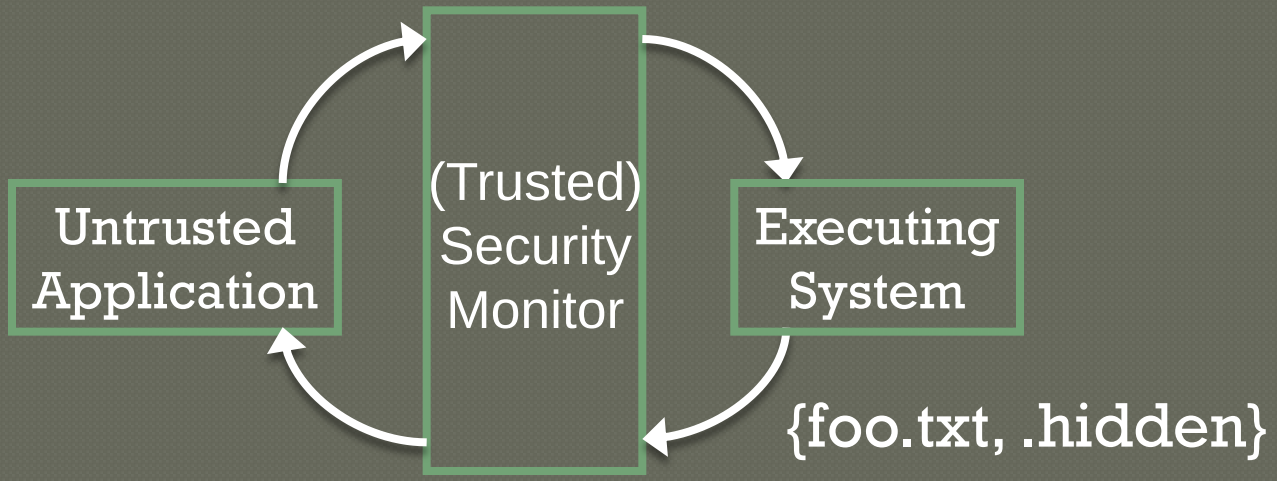
Example Execution

- ls_i ; ls_o ; $\{foo.txt, .hidden\}_i$; $\{foo.txt\}_o$



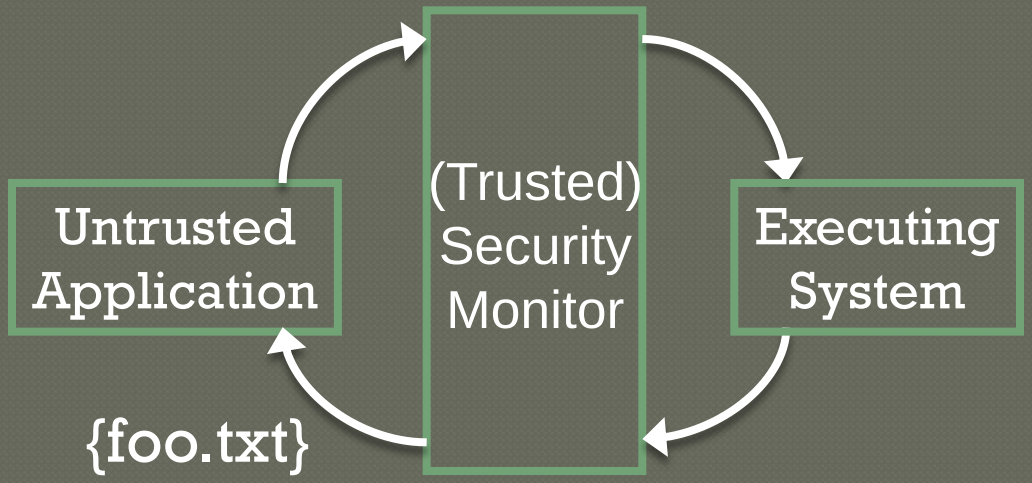
Example Execution

- ls_i ; ls_o ; $\{foo.txt, .hidden\}_i$; $\{foo.txt\}_o$



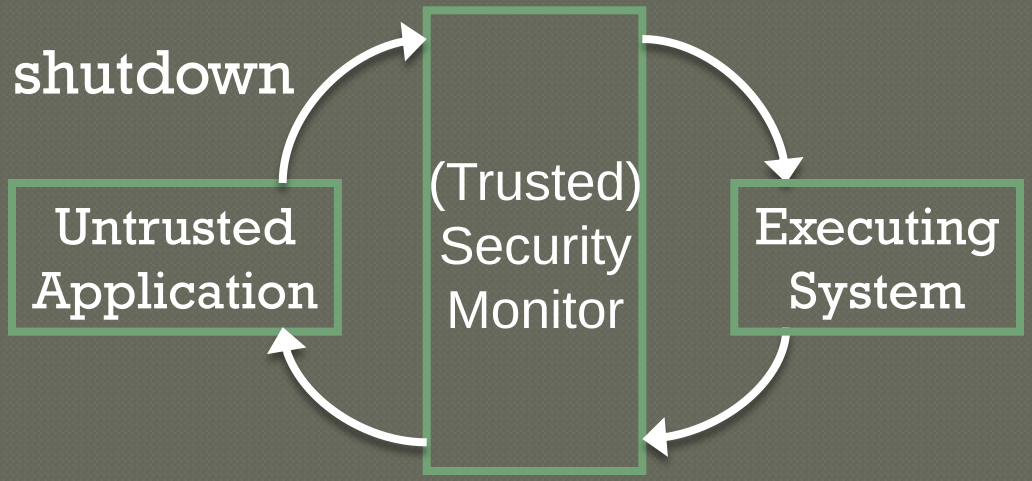
Example Execution

- ls_i ; ls_o ; $\{foo.txt, .hidden\}_i$; $\{foo.txt\}_o$



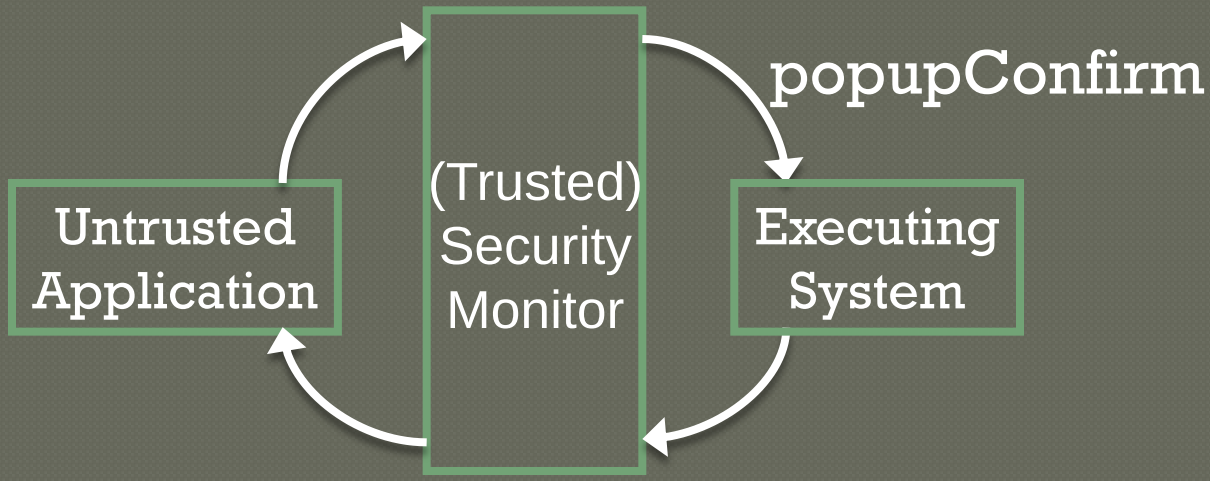
Another Example Execution

- $shutdown_i$; $popupConfirm_o$; OK_i ; $shutdown_o$



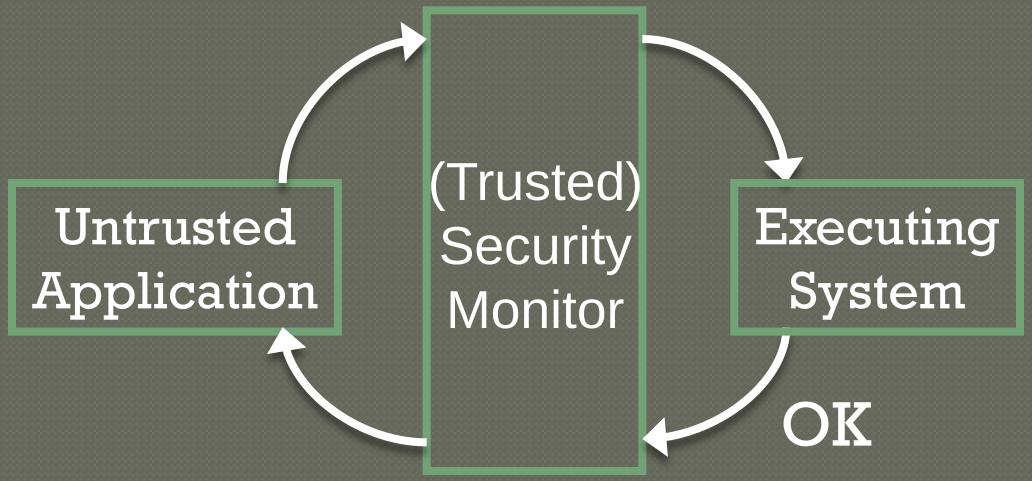
Another Example Execution

- $shutdown_i$; $popupConfirm_o$; OK_i ; $shutdown_o$



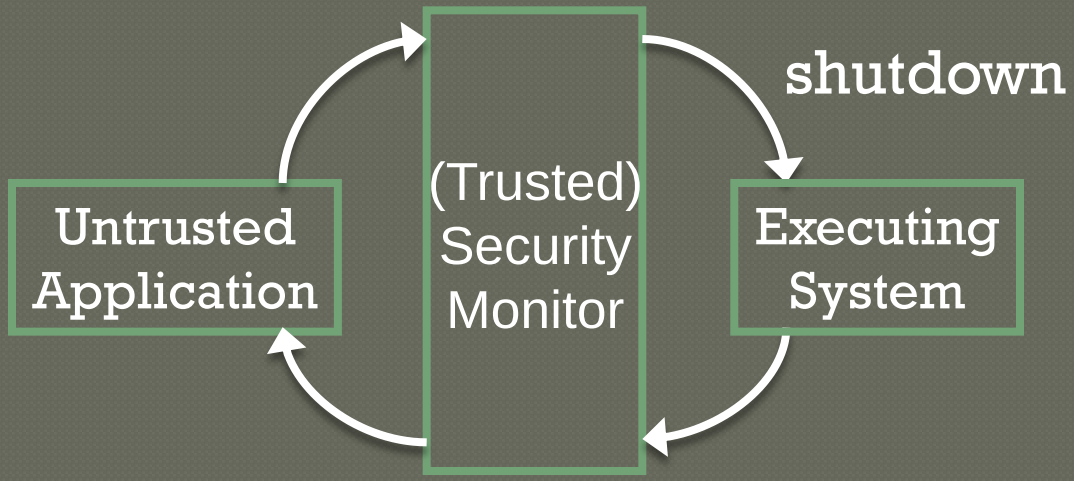
Another Example Execution

- $shutdown_i$; $popupConfirm_o$; OK_i ; $shutdown_o$



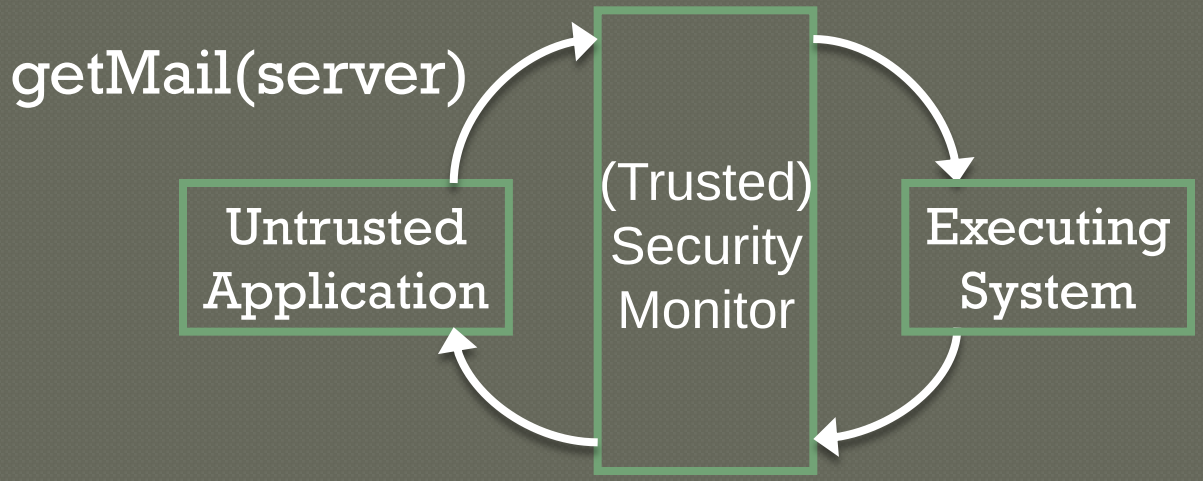
Another Example Execution

- $shutdown_i$; $popupConfirm_o$; OK_i ; $shutdown_o$



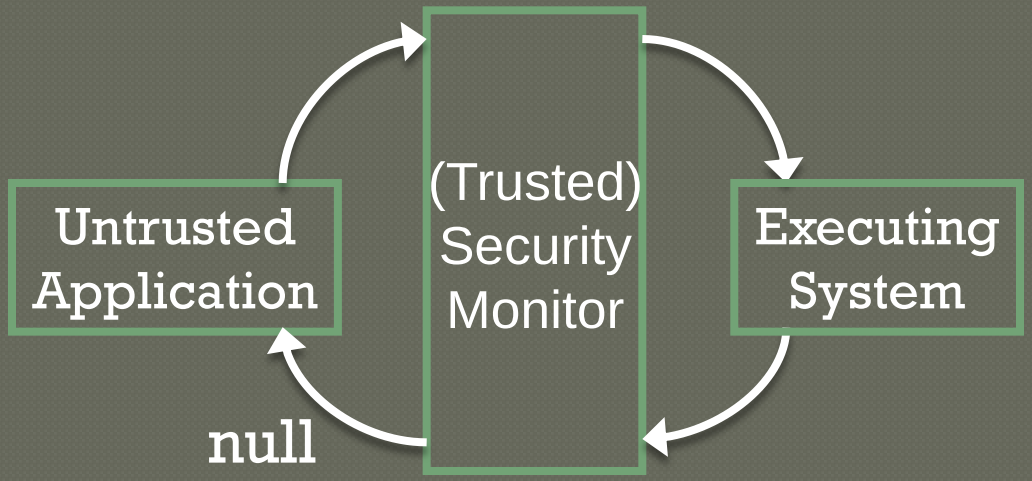
Last Example Execution

- getMail(server)_i ; null_o ; getMail(server)_i ; null_o ; …



Last Example Execution

- getMail(server)_i ; null_o ; getMail(server)_i ; null_o ; …



Last Example Execution

- getMail(server)_i ; null_o ; getMail(server)_i ; null_o ; …

Etc… This is an infinite-length execution, so it represents a nonterminating run of the monitor (and target application)

Notation

- E^* = set of all well-formed finite-length executions on system with event set E
- E^ω = set of all well-formed infinite-length executions on system with event set E
- $E^\infty = E^* \cup E^\omega$
- $\bullet$ = empty execution (no events occur)
- $x;x'$ = well-formed concatenation of executions x and x'
- $x \leq x'$ = execution x is a prefix of x'

More Notation

- Metavariable ____ ranges over ____
  - e over events
  - a over actions
  - r over results
  - x over executions
  - α over $A \cup \{\bullet\}$ (potential actions)
  - ρ over $R \cup \{\bullet\}$ (potential results)

Definition of MRAs

- An MRA M is a tuple (E, Q, q_0, δ)
  - E = event set over which M operates
  - Q = M's finite or countably infinite state set
  - q_0 = M's initial state
  - δ = M's (partially recursive) transition function

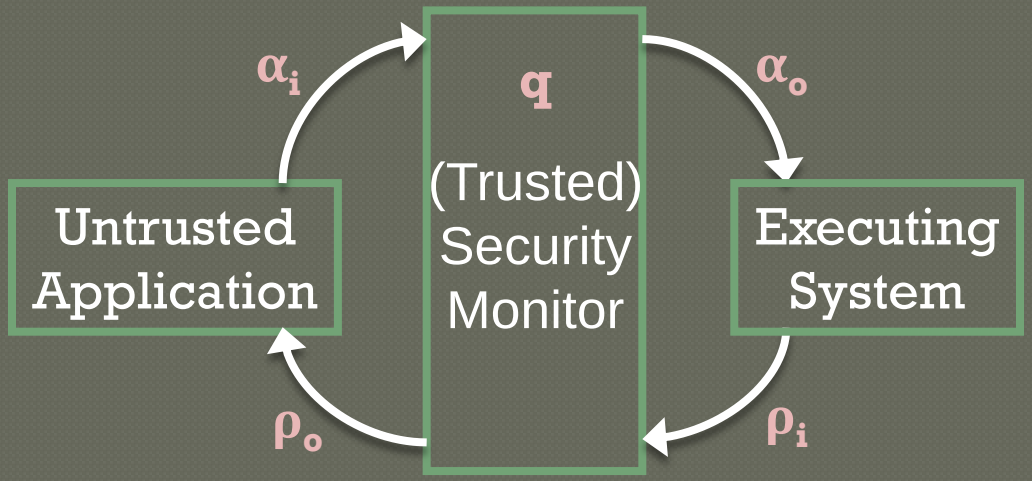
$$\delta : Q \times E \rightarrow Q \times E$$

given a current MRA state and an event just input,
δ returns the next MRA state and an event to output

MRA *Configurations*

$$\begin{array}{c|c|c} \alpha_i & & \alpha_o \\ \rho_o & q & \rho_i \end{array}$$

- q is the MRA's current state
- α_i is empty or the action being input to the MRA
- α_o is empty or the action being output from the MRA
- ρ_i is empty or the result being input to the MRA
- ρ_o is empty or the result being output from the MRA



MRA Operational Semantics

- Starting configuration: $|q_0|$

- A single-step judgment specifies how MRAs take small steps (to input/output a single event)
  - Single-step judgment form: $C \xrightarrow{e} C'$

- Then the multi-step judgment is the reflexive, transitive closure of the single-step relation
  - Multi-step judgment form: $C \xrightarrow{x}{}^{*} C'$

Single-step Rules

- Rules for inputting and reacting to *actions*:

$$\frac{next_T = a}{\rho\left|\mathbf{q}\right| \xrightarrow{a_i} {}^{a}\left|\mathbf{q}\right|}$$

(Input-Action)

$$\frac{\delta(q,a) = (q',a')}{{}^{a}\left|\mathbf{q}\right| \xrightarrow{a'_o} \left|\mathbf{q'}\right|^{a'}}$$

(Output-Action-for-Action)

$$\frac{\delta(q,a) = (q',r)}{{}^{a}\left|\mathbf{q}\right| \xrightarrow{r_o} {}_{r}\left|\mathbf{q'}\right|}$$

(Output-Result-for-Action)

Single-step Rules

- Rules for inputting and reacting to *results*:

$$\frac{\text{next}_S = r}{\left|\mathbf{q}\right|^a \xrightarrow{r_i} \left|\mathbf{q}\right|_r} \quad \text{(Input-Result)}$$

$$\frac{\delta(q,r) = (q',a)}{\left|\mathbf{q}\right|_r \xrightarrow{a_o} \left|\mathbf{q'}\right|^a} \quad \text{(Output-Action-for-Result)}$$

$$\frac{\delta(q,r) = (q',r')}{\left|\mathbf{q}\right|_r \xrightarrow{r'_o} {}_{r'}\left|\mathbf{q'}\right|} \quad \text{(Output-Result-for-Result)}$$

One More Operational Judgment

- $M \Downarrow x$ means MRA M, when its input events match the (possibly infinite) sequence of input events in x, *produces* the execution x
  - $M \Downarrow x$ iff:
    - if $x \in E^{\omega}$ then $\forall\, x' \leq x : \exists C : C_0 \xrightarrow{x'}{}^{*} C$
    - if $x \in E^{*}$ then $\exists C$:
      - $C_0 \xrightarrow{x}{}^{*} C$
      - if x ends with an input event then M never transitions from C

Observation

- Semantics matches the possible behaviors we've observed in many implemented monitoring systems
  - Polymer (with Bauer and Walker)
  - PSLang (Erlingsson and Schneider)
  - AspectJ (Kiczales et al.)
  - Etc.

Example MRA

- Hidden-file filtering MRA M = (E, Q, q_0, δ)
  - E = { ls, …}
  - Q = { T , F } (are we executing an ls?)
  - q_0 = { F }
  - $\delta(q,e) = \begin{cases} (F, e) & \text{if } q=F \text{ and } e<>ls \\ (T, e) & \text{if } q=F \text{ and } e=ls \\ (F, filter(e)) & \text{if } q=T \end{cases}$

Another Example MRA

- Shutdown-confirming MRA M=(E, Q, q_0, δ)
  - E = { shutdown, popupConfirm, OK, cancel, null, …}
  - Q = { T , F } (are we confirming a shutdown?)
  - q_0 = { F }

$$\delta(q,e) = \begin{cases} (F, e) & \text{if } q=F \text{ and } e<>\text{shutdown} \\ (T, \text{popupConfirm}) & \text{if } q=F \text{ and } e=\text{shutdown} \\ (F, \text{null}) & \text{if } q=T \text{ and } e=\text{cancel} \\ (F, \text{shutdown}) & \text{if } q=T \text{ and } e=\text{OK} \end{cases}$$

Definition of Policies

- (Technical note: here we're really only considering special kinds of policies called *properties*)
- Policies are predicates on (or sets of) executions
- P(x) iff execution x satisfies policy P

Example: Definition of the Filter-hidden-files Policy

$P(\bullet)$

[it's OK for the target to do nothing]

$\neg P(ls_i)$

[monitor may not just stop upon inputting ls; must then output ls]

$P(ls_i ; e_o)$ iff $e=ls$

[monitor must output only ls after inputting ls; it's then OK for the system to never return a listing]

$\forall$ results L:
$\neg P(ls_i ; ls_o ; L_i)$

[monitor may not stop upon inputting L; must return the filtered list to the target]

$P(ls_i ; ls_o ; L_i ; e_o)$ iff $e=filter(L)$

[monitor must filter listings]

How Policies in MRA Model Differ from Those of Previous Models

- Policies here can reason about results
  - Enables result-sanitization policies
  - E.g., filter-hidden-file policy
- Policies here can reason about *input* events
  - Enables policies to dictate exactly how mechanisms can/must transform events
  - E.g., confirm-shutdown policy

=> Powerful, but practical, expressiveness

Definitions of Enforcement

- Sound enforcement (no false -s)

  MRA M *soundly* enforces policy P iff
  $\forall x \in E^{\infty}: (M \Downarrow x \Rightarrow P(x))$

- Complete enforcement (no false +s)

  MRA M *completely* enforces policy P iff
  $\forall x \in E^{\infty}: (P(x) \Rightarrow M \Downarrow x)$

- Precise enforcement (no false +s or -s)

  MRA M *precisely* enforces policy P iff
  $\forall x \in E^{\infty}: (M \Downarrow x \Leftrightarrow P(x))$

How Enforcement in MRA Model Differs from That of Previous Models

- Simpler: no need for extra "*transparency*" constraints that can be rolled into policy definitions (now that policies can reason about input events)
- More expressive: can reason about *complete* and *precise* enforcement too

Outline

- Research questions
  - How do monitors operate to enforce policies?
    - Which policies can runtime mechanisms enforce?
- Related work vs. this work
- The model: systems, executions, monitors, policies, and enforcement
- Analysis of enforceable properties
  - What are the limits of MRA enforcement?
- Summary and future work

Sound Enforcement with MRAs

- Policy P on system with event set E can be soundly enforced by some MRA M iff there exists (R.E.) predicate R over E^* s.t. all the following are true.
  - $R(\bullet)$
  - $\forall(\mathbf{x};e_i)\in E^*$:
    - $\neg R(\mathbf{x})$ or
    - $P(\mathbf{x};e_i)$ or
    - $\exists e'\in E:(R(\mathbf{x};e_i;e'_o) \wedge P(\mathbf{x};e_i;e'_o))$
  - $\forall \mathbf{x}\in E^\omega$: if $\neg P(\mathbf{x})$ then $\exists(\mathbf{x}';e_i)\leq \mathbf{x}:\neg R(\mathbf{x}')$

Complete Enforcement with MRAs

- Policy P on system with event set E can be completely enforced by some MRA M iff:
  - $\forall(x;e_i)\in E^*$:
    - $\forall e'\in E : dead_P(x;e_i;e'_o)$ or
    - $\neg P(x;e_i) \wedge \exists! e'\in E : alive_P(x;e_i;e'_o)$

(where $alive_P(x)$ iff $\exists x'\in E^\infty : P(x;x')$
and $dead_P(x)$ iff $\neg alive_P(x)$)

Precise Enforcement with MRAs

- Policy P on system with event set E can be precisely enforced by some MRA M iff all the following are true.
  - $P(\bullet)$
  - $\forall(x;e_i)\in E^*$:
    - $\neg P(x)$ or
    - $P(x;e_i) \wedge \forall e' \in E : dead_P(x;e_i;e'_o)$ or
    - $\neg P(x;e_i) \wedge \exists! e' \in E : P(x;e_i;e'_o) \wedge \exists! e' \in E : alive_P(x;e_i;e'_o)$
  - $\forall x \in E^\omega$: if $\neg P(x)$ then $\exists(x';e_i) \leq x : \neg P(x')$

Outline

- Research questions
  - How do monitors operate to enforce policies?
    - Which policies can runtime mechanisms enforce?
- Related work vs. this work
- The model: systems, executions, monitors, policies, and enforcement
- Analysis of enforceable properties
- Summary and future work

Summary

- Started building a theory of runtime enforcement based on MRAs, which:
  - model the realistic ability of runtime mechanisms to transform synchronous actions and their results.
  - can enforce result-sanitization policies and policies based on input events.
  - provide simpler and more expressive definitions of *policies* and *enforcement* than previous models.

Future Work

- Something between edit automata (which assume asynchronous actions) and MRAs (which assume synchronous actions)?
  - How would the monitor know when the target is waiting for a result, and for which action?
    - Static analysis of target application?
    - Could get complicated

Additional Future Work

- Which policies get enforced when we combine runtime mechanisms?
- How efficiently does a mechanism enforce a policy?
- What are the lower bounds on resources required to enforce policies of interest?

- Having a realistic operational model of runtime enforcement seems like a good first step to address these research questions

Thanks/Questions?