A Theory of Runtime Enforcement, with Results

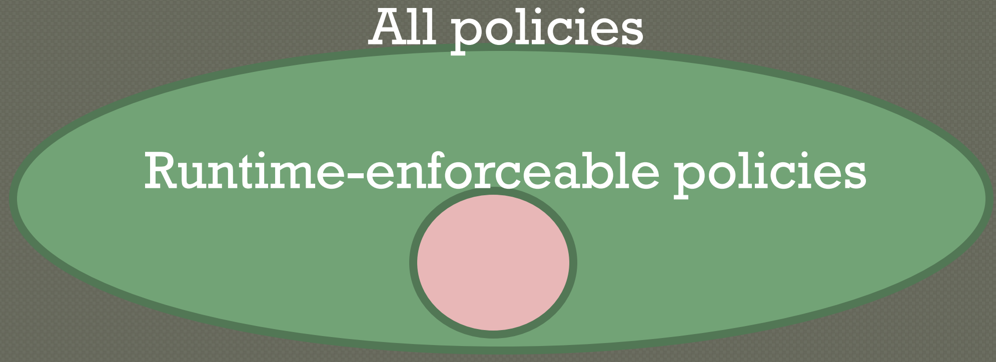
Jay Ligatti and Srikar Reddy
University of South Florida

Runtime Mechanisms

- Also known as runtime/security/program *monitors*
- 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.

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

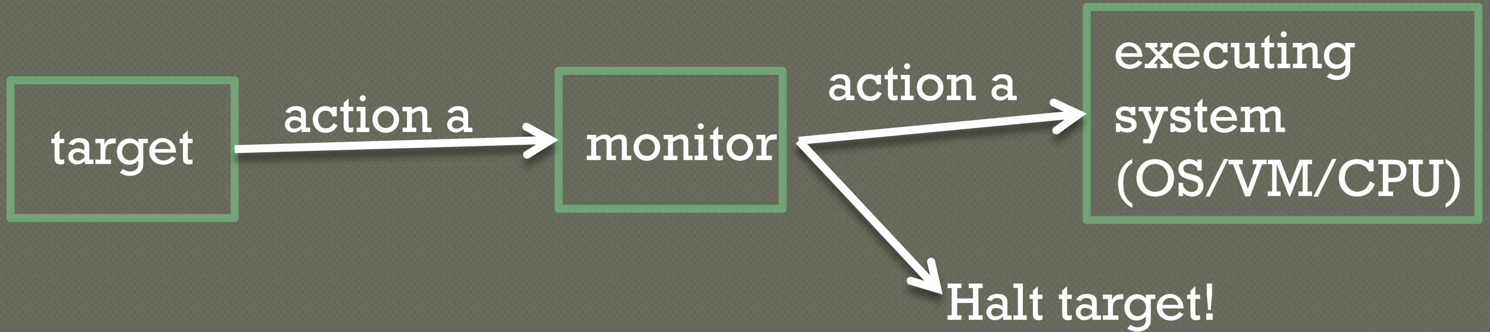
- How do monitors operate to enforce policies?
  - Which policies can runtime mechanisms enforce?
  - Which policies get enforced when we combine runtime mechanisms?
  - How efficiently does a mechanism enforce a policy?
  - What are the lower bounds on resources required to enforce policies of interest?

Outline

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

Related Work: Truncation Automata

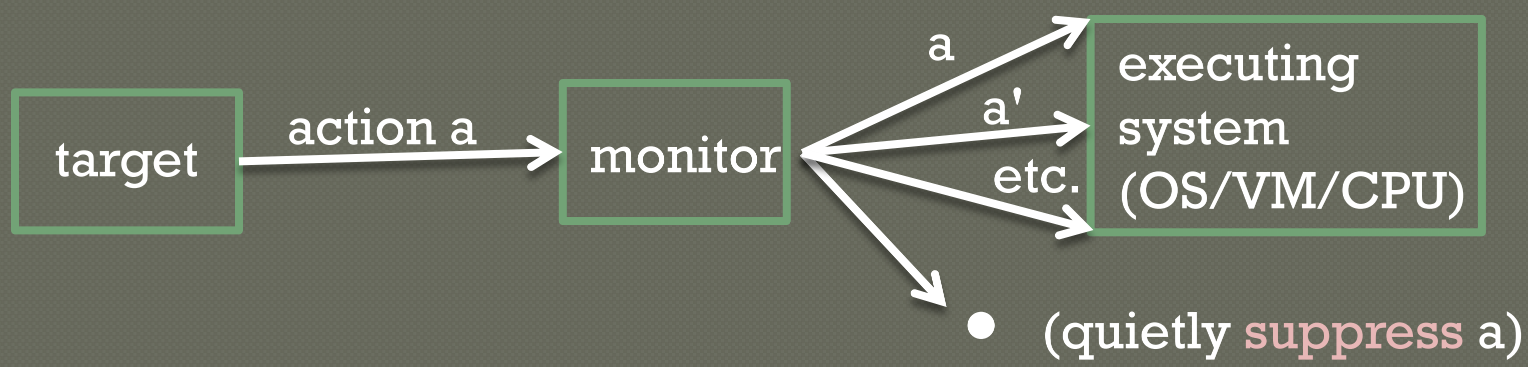
- Most analyses of monitors are based on truncation automata (Schneider, 2000)



- 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



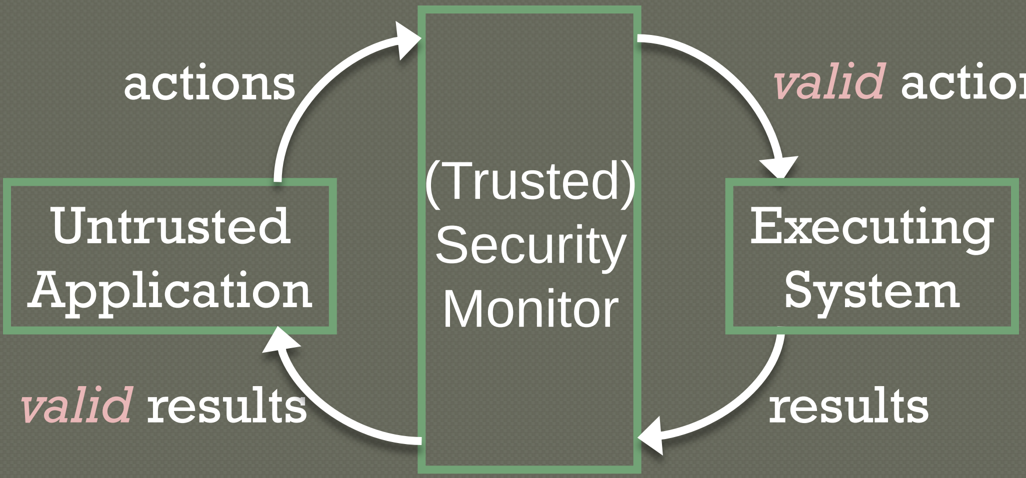
- Operation: actively transform target actions to ensure they satisfy desired policy

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) (or *Synchronous* Edit Automata (SEAs))

- Conservatively assume all actions are synchronous



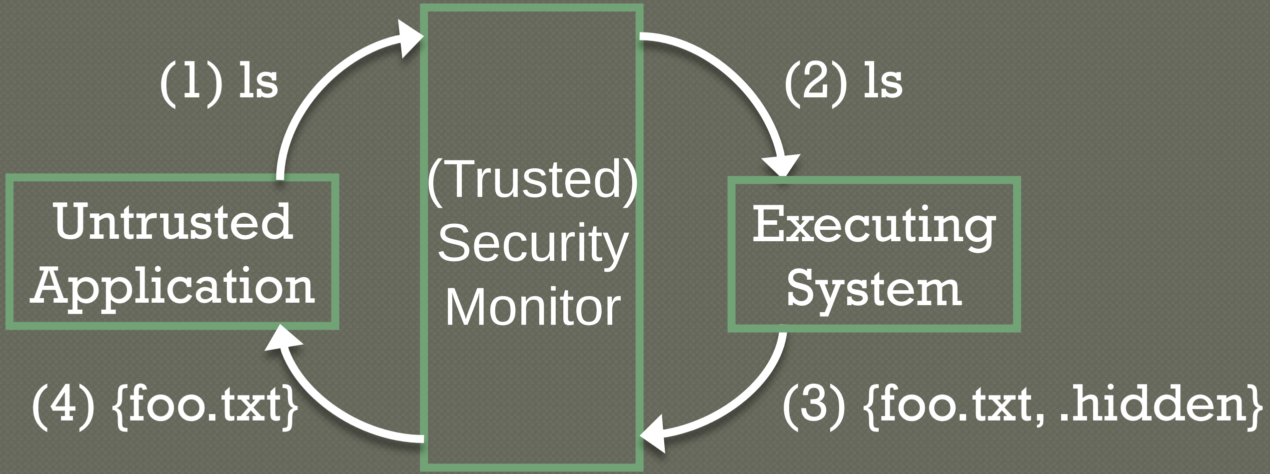
- Operation: actively transform target actions and results of those actions to ensure they satisfy desired policy

This Work: Mandatory Results Automata (MRAs) (or *Synchronous* Edit Automata (SEAs))

- 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)

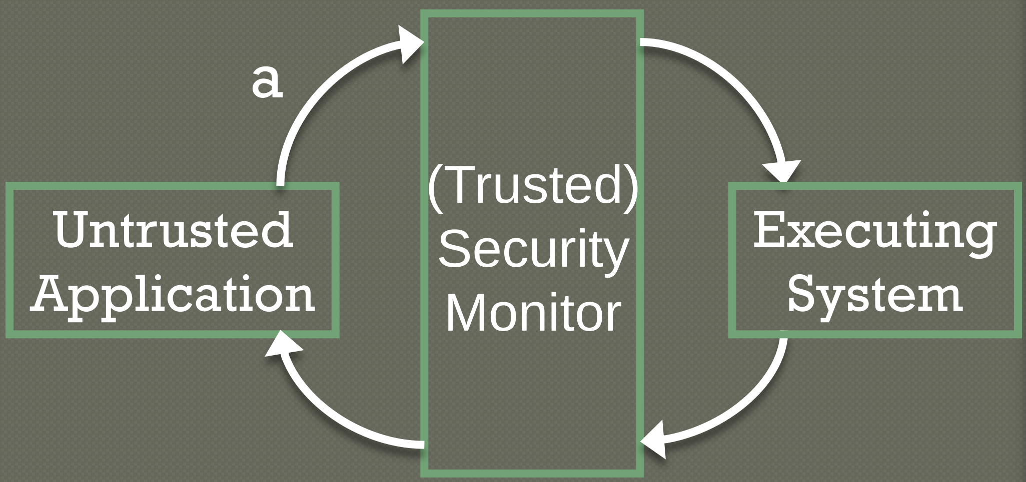
Outline

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

Definition of MRA traces/executions

- Execution: finite or countably infinite sequence of MRA-relevant events (i.e., *actions* and *results*)

- 4 possibilities:
  (1) MRA *inputs* action *a* from the target

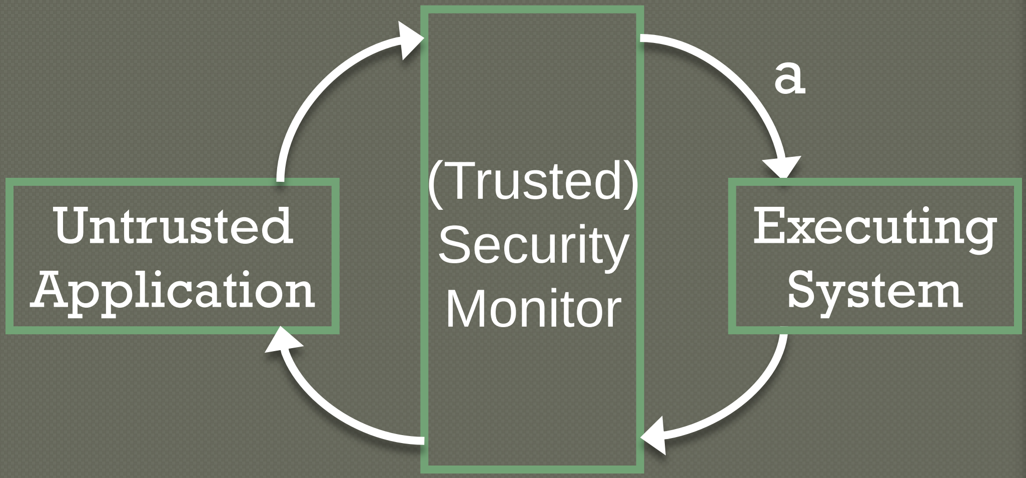


=> add a_i to the current trace

Definition of MRA traces/executions

- Execution: finite or countably infinite sequence of MRA-relevant events (i.e., *actions* and *results*)

- 4 possibilities:
  (2) MRA *outputs* action *a* to be executed

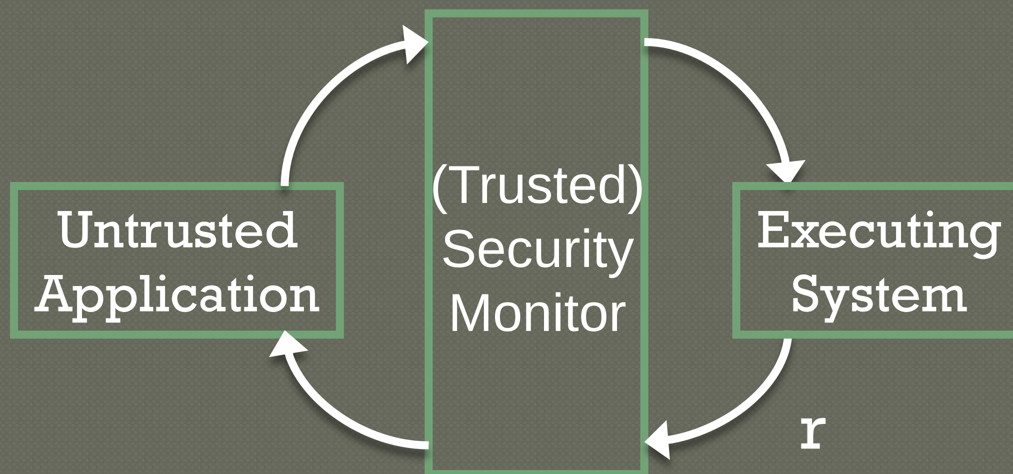


=> add a_o to the current trace

Definition of MRA traces/executions

- Execution: finite or countably infinite sequence of MRA-relevant events (i.e., *actions* and *results*)

- 4 possibilities:
  (3) MRA *inputs* result *r* from the system

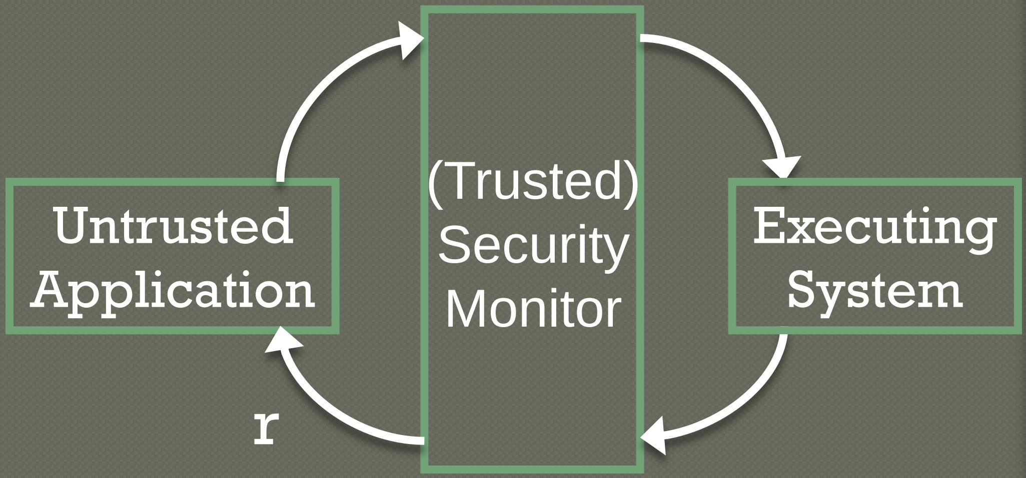


=> add r_i to the current trace

Definition of MRA traces/executions

- Execution: finite or countably infinite sequence of MRA-relevant events (i.e., *actions* and *results*)

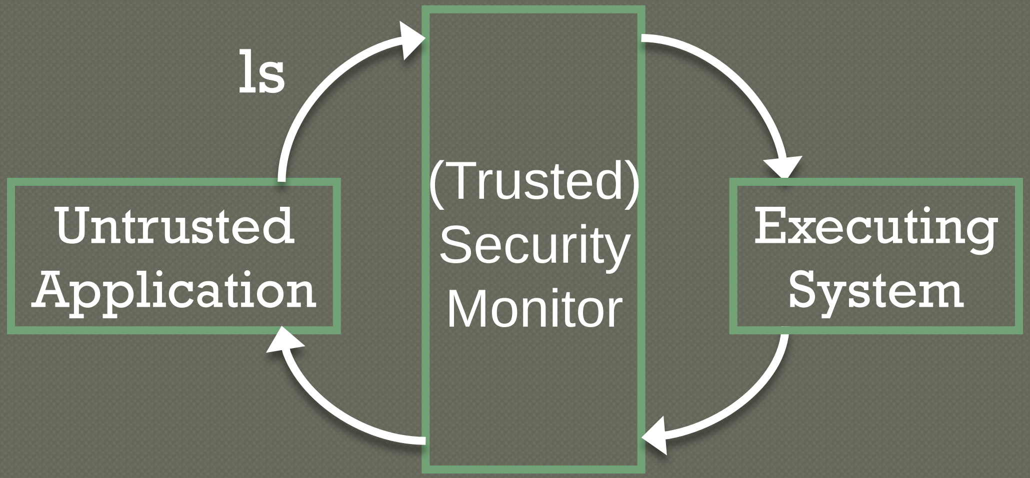
- 4 possibilities:
  (4) MRA *outputs* result r to the target



=> add r_o to the current trace

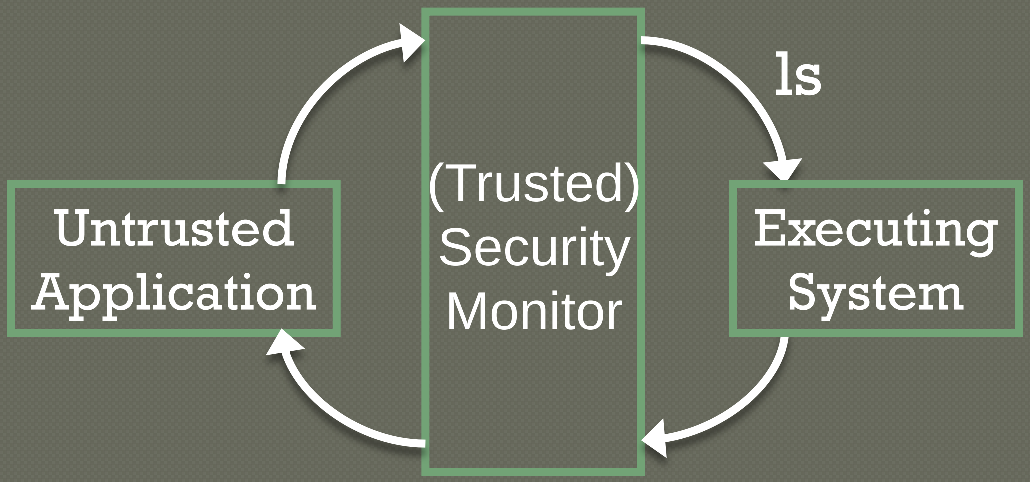
Example Execution

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



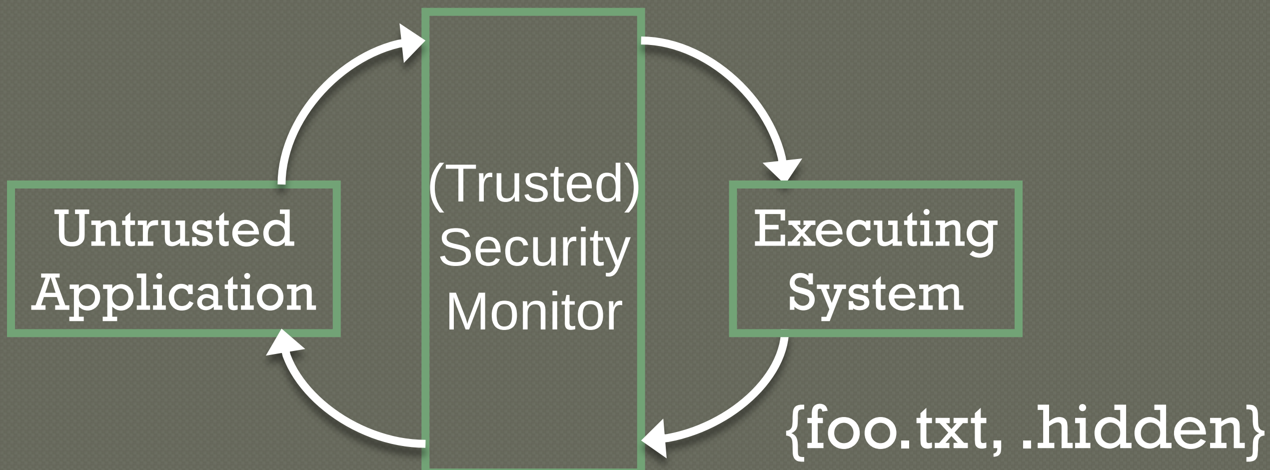
Example Execution

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



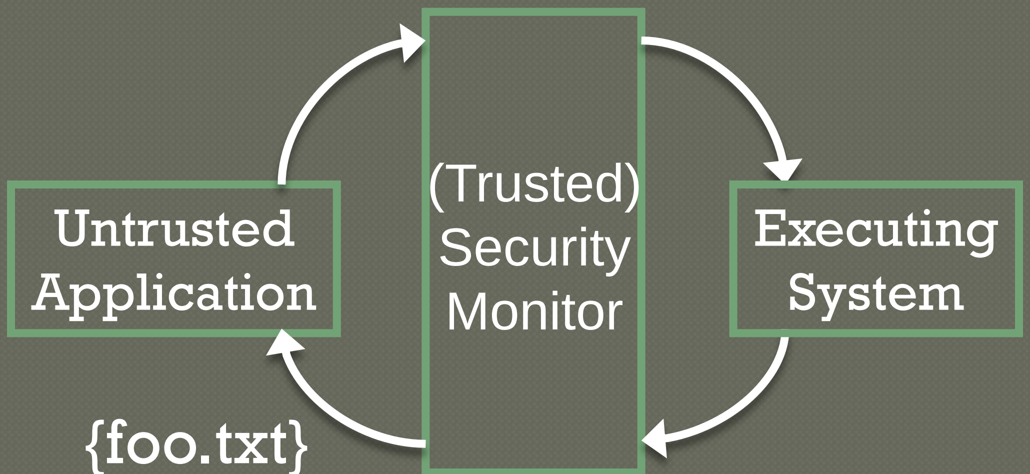
Example Execution

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



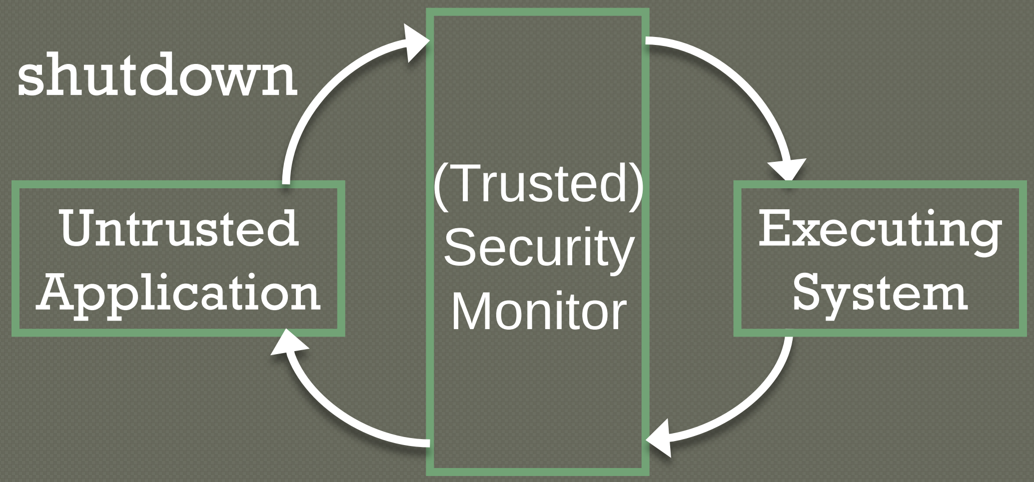
Example Execution

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



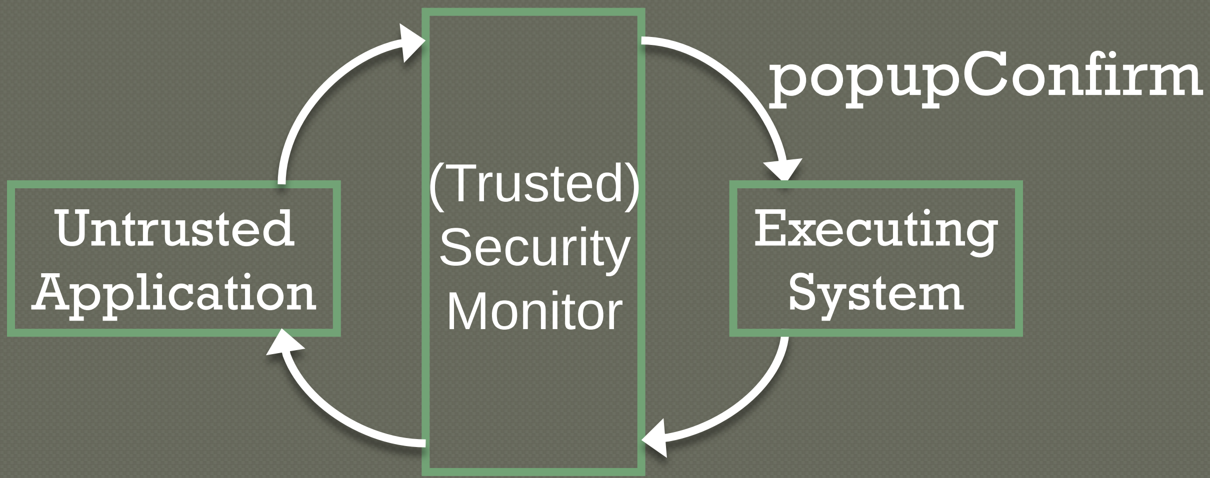
Another Example Execution

- shutdown$_i$; popupConfirm$_o$; OK$_i$; shutdown$_o$



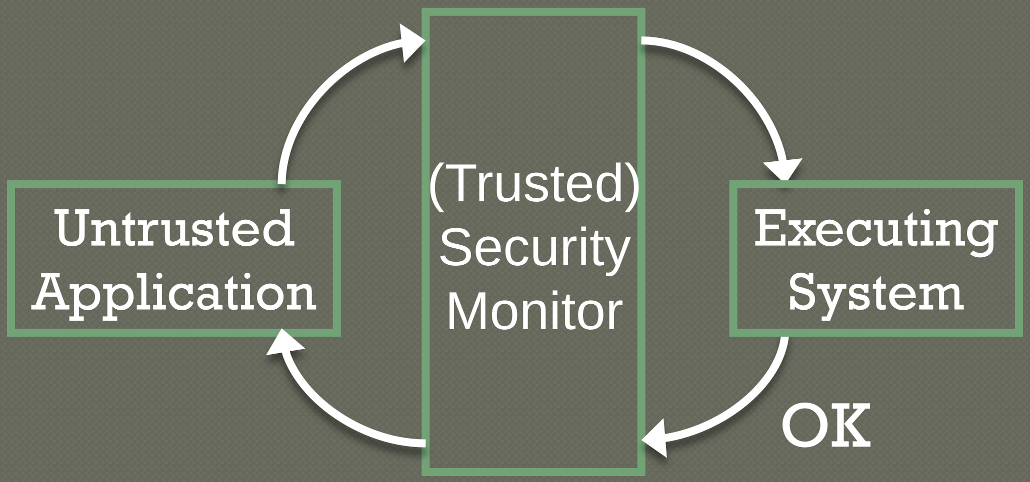
Another Example Execution

- shutdown_i ; popupConfirm_o ; OK_i ; shutdown_o



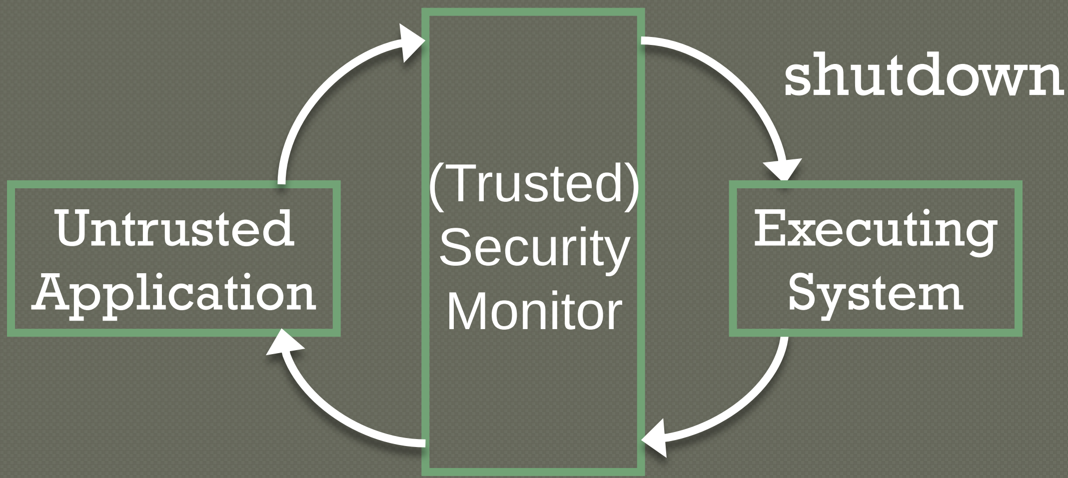
Another Example Execution

- shutdown_i ; popupConfirm_o ; OK_i ; shutdown_o



Another Example Execution

- shutdown_i ; popupConfirm_o ; OK_i ; shutdown_o



Definition of MRAs

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

$$\partial : Q \times E \rightarrow Q \times E$$

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

Example MRA

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

Another Example MRA

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

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

Observation

- MRA operations match 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.
- For every input action and input result, monitor may output an action or a result
- Previous models couldn't transform results => couldn't model the last 2 realistic examples

MRA Operational Semantics

- MRA operations can be formalized with six small rules dictating how traces get built

$$\frac{next_T = a}{{}_{\rho}\left|q\right| \xrightarrow{a_i} {}^{a}\left|q\right|}\ \textit{(Input-Action)} \qquad \frac{next_S = r}{\left|q\right|^{a} \xrightarrow{r_i} \left|q\right|_{r}}\ \textit{(Input-Result)}$$

$$\frac{\delta(q,a) = (q',a')}{{}^{a}\left|q\right| \xrightarrow{a'_o} \left|q'\right|^{a'}}\ \textit{(Output-Act-for-Act)} \qquad \frac{\delta(q,r) = (q',a)}{\left|q\right|_{r} \xrightarrow{a_o} \left|q'\right|^{a}}\ \textit{(Output-Act-for-Res)}$$

$$\frac{\delta(q,a) = (q',r)}{{}^{a}\left|q\right| \xrightarrow{r_o} {}_{r}\left|q'\right|}\ \textit{(Output-Res-for-Act)} \qquad \frac{\delta(q,r) = (q',r')}{\left|q\right|_{r} \xrightarrow{r'_o} {}_{r'}\left|q'\right|}\ \textit{(Output-Res-for-Res)}$$

Fig. 2. Single-step semantics of mandatory results automata.

- Please see conference proceedings for details

Definition of Policies

- (Technical note: here we're really only considering special kinds of policies called properties)
- Policies are predicates on 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]

$\forall$ directory listings 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)

  M *soundly* enforces P iff
  $\forall$ executions x: (M produces x $\Rightarrow$ P(x))

- Complete enforcement (no false +s)

  M *completely* enforces P iff
  $\forall$ executions x: (P(x) $\Rightarrow$ M produces x)

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

  M *precisely* enforces policy P iff
  M soundly and completely enforces P

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: executions, monitors, policies, and enforcement
- Analysis of enforceable properties
- Summary and future work

Sound Enforcement of Properties with MRAs

Theorem 1. *Property $\hat{P}$ on a system with event set E can be* **soundly** *enforced by some MRA M iff there exists recursively enumerable predicate R over E^* such that all the following are true.*

1. $R(\cdot)$
2. $\forall (x; e_i) \in E^* : \left(\neg R(x) \;\vee\; \hat{P}(x; e_i) \;\vee\; \exists e' \in E : \left(\begin{array}{r} R(x; e_i; e'_o) \\ \wedge\; \hat{P}(x; e_i; e'_o) \end{array} \right) \right)$
3. $\forall x \in E^\omega : \left(\neg \hat{P}(x) \implies \exists (x'; e_i) \preceq x : \neg R(x') \right)$

Complete Enforcement of Properties with MRAs

Theorem 2. *Property $\hat{P}$ on a system with event set E can be* **completely** *enforced by some MRA M iff:*

$$\forall (x; e_i) \in E^* : \begin{pmatrix} \forall e' \in E : dead(x; e_i; e'_o) \\ \vee \neg \hat{P}(x; e_i) \;\wedge\; \exists_1 e' \in E : alive(x; e_i; e'_o) \end{pmatrix}$$

Precise Enforcement of Properties with MRAs

Theorem 3. *Property $\hat{P}$ on a system with event set E can be* **precisely** *enforced by some MRA M iff all the following are true.*

1. $\hat{P}(\cdot)$

2. $\forall (x; e_i) \in E^* : \left(\begin{array}{l} \quad \neg\hat{P}(x) \\ \vee \ \hat{P}(x; e_i) \ \wedge \ \forall e' \in E : dead(x; e_i; e'_o) \\ \vee \ \neg\hat{P}(x; e_i) \ \wedge \ \exists_1 e' \in E : \hat{P}(x; e_i; e'_o) \\ \qquad\qquad\qquad \wedge \ \exists_1 e' \in E : alive(x; e_i; e'_o) \end{array} \right)$

3. $\forall x \in E^\omega : \left(\neg\hat{P}(x) \implies \exists (x'; e_i) \preceq x : \neg\hat{P}(x') \right)$

Outline

- Research questions
  - How do monitors operate to enforce policies?
    - Which policies can runtime mechanisms enforce?
- Related work vs. this work
- The model: 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?