STRL

Deriving Enforcement Mechanisms from Policies

Helge Janicke, Antonio Cau, François Siewe, Hussein Zedan

Software Technology Research Laboratory
De Montfort University

Policy 2007, 14th June 2007, in Bologna, Italy

Motivation

STRL

- Policies describe protection requirements in an abstract, often denotational form.
- In security critical applications an unambiguous and concise semantics of policies is required.
- Abstract policies must be translated (interpreted) and enforced.

- How to ensure that enforcement mechanisms are *correct*?
- Can we accurately define what *correct* means?
- What optimisation of the enforcement is possible?
- Is the approach constructive and can it be automated?

STRL

Expressions

$$e \quad ::= \quad \mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \bigcirc v \mid \text{fin } v$$

Formulae

$$f \quad ::= \quad p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \text{skip} \mid f_1 \; ; \; f_2 \mid f^*$$

- μ is an integer value,
- a is a static variable (doesn't change within an interval),
- A is a state variable (can change within an interval),
- v is a static or state variable,
- g is a function symbol and
- p is a predicate symbol

Motivation
ITL
Policy Rules
Enforcement
Summary

Expressions

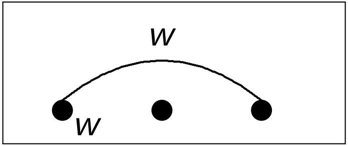
$$e \quad ::= \quad \mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \bigcirc v \mid \text{fin } v$$

Formulae

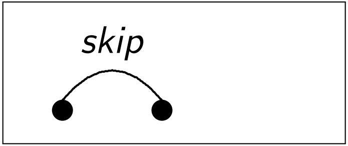
$$f \quad ::= \quad p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \text{skip} \mid f_1 \,;\, f_2 \mid f^*$$

- μ is an integer value,
- a is a static variable (doesn't change within an interval),
- A is a state variable (can change within an interval),
- v is a static or state variable,
- g is a function symbol and
- p is a predicate symbol

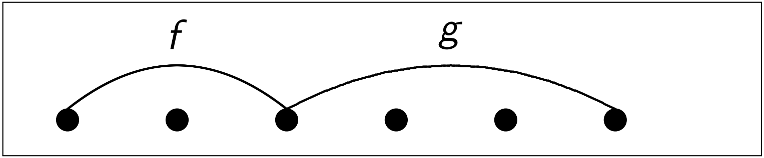
State Formula



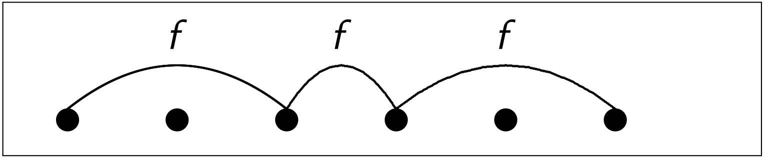
Skip (Unit Interval)



Chop (Sequence)



Chopstar (Iteration)



Policy Rule

Policy Rule

Expresses individual protection requirements in the form:

$$\textit{premise} \longrightarrow \textit{consequence}$$

- **Premise** describes the behaviour (as an ITL formula) that leads to the consequence.

  *"Subject **S** did in the past **read** object **O**"*

- **Consequence** distinguishes the type of the rule.

  *"then **S** is authorised to **read** objects from the same dataset"*

Semantics of Rules

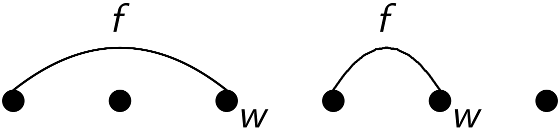
Definition (Always Followed By)

The operator *always-followed-by*, is defined as:

$$f \mapsto w \mathrel{\widehat{=}} \boxdot((\Diamond f) \supset fin\ w)$$

where f stands for any ITL formula, and w is a state formula.

$f \mapsto w$



A policy defines access control decisions $autho(s, o, a)$ in each state of the interval.

We define the execution of requests such that:

- *done(s,o,a)* is true iff the action was successful.
- *failed(s,o,a)* is true iff the action failed.

Definition (Correct Enforcement — Access Control)

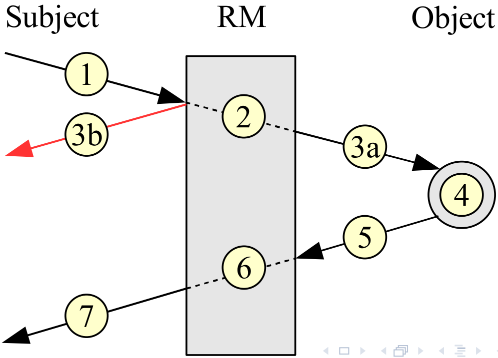
We say a policy is *correctly* enforced iff:

$$E_{autho} \mathrel{\widehat{=}} \text{keep}\,(\bigcirc done(s, o, a) \supset autho(s, o, a))$$

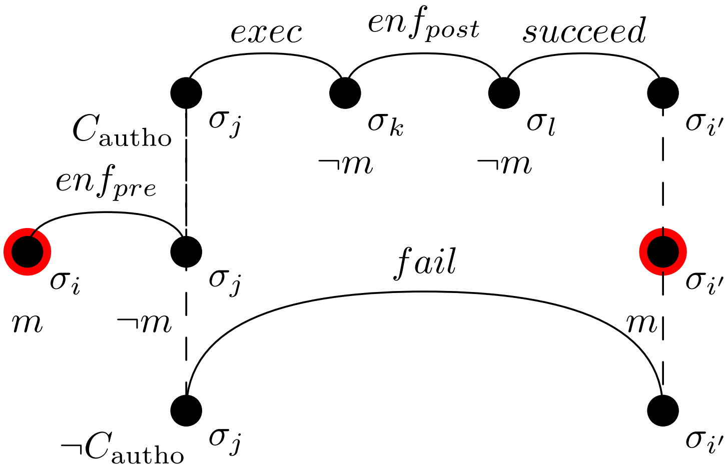
STRL

Rules define *history-based* access control. Their enforcement must:

- Determine the history that is required for policy decisions.
- Maintain this history.
- Optimise enforcement efficiency and decide timely.

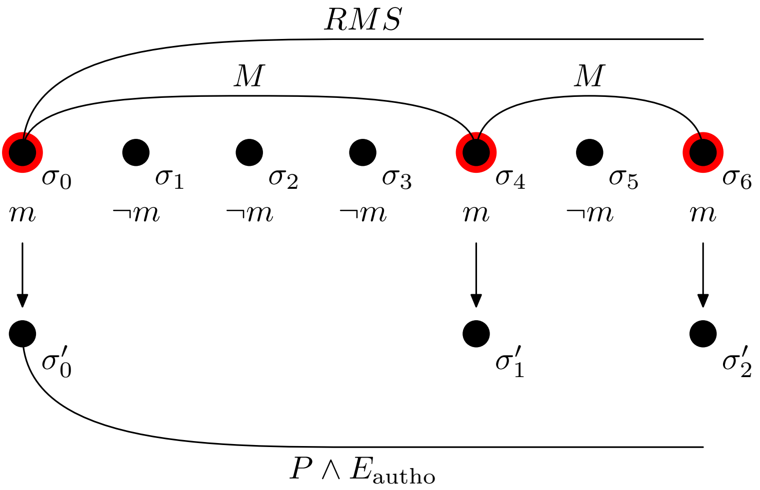


STRL



Requests are defined at fine level of temporal granularity. Policy enforcement takes place in enf_{pre} and enf_{post} and is reflected in the condition C_{autho}.

Enforcement
Mapping between Policies and Enforcement



We use *temporal projection* to map between the more coarse policy reference interval and the fine grained RM specification.

Enforcement
A Simple Rule

Subject s is authorised to perform a on o if s was not acting in the role admin in the state before.

$$1 : \neg\, in(s, admin) \;\mapsto\; autho(s, o, a)$$

We stepwise refine the temporal operators. It is clear that only the current and the last value of the role assignments are required. This allows to refine the pre-update as.

$$enf_{pre} \mathrel{\widehat{=}} \forall s \in \mathcal{S} \bullet$$
$$H_{\mathrm{in,s,admin}}[1], H_{\mathrm{in,s,admin}}[0] \leftarrow H_{\mathrm{in,s,admin}}[0], \mathrm{in}(s, \mathrm{admin})$$

where H is a list of history variables for the observed subscript.

The (parallel) temporal assignment can be refined into the following sequence:

$$enf_{pre} \mathrel{\widehat{=}} \text{for } s \text{ in } \mathcal{S} : \{ \quad H_{\mathrm{in,s,admin}}[1] := H_{\mathrm{in,s,admin}}[0]; \quad H_{\mathrm{in,s,admin}}[0] := \mathrm{in}(s, \mathrm{admin}) \quad \}$$

As the relevant history is now available, we can express the actual access decision in terms of these variables.

$$C_{autho} \mathrel{\widehat{=}} T \geq 1 \wedge \neg H_{\mathrm{in,s,admin}}[1]$$

Summary

STRL

- Policies define history-based access control decisions at an abstract level.
- Enforcement defines the concrete mechanism behaviour at a very concrete level of abstraction.
- We use temporal projection to map between this level.
- Correctness of the enforcement is defined as a property on this mapping.
- The different abstraction levels allow for the introduction of states that define code required for the maintenance of a history.
- This code can be derived from the high-level policy specification.
- The formal underpinning allows for (correctness preserving) optimisations.

Thank you for your Questions and Comments!

Contact:

Helge Janicke (heljanic@dmu.ac.uk)

Abbreviations

$\bigcirc f$	$\widehat{=}$	skip ; f
more	$\widehat{=}$	skip ; true
empty	$\widehat{=}$	$\neg$ more
inf	$\widehat{=}$	true ; false
finite	$\widehat{=}$	$\neg$ inf
$\Diamond f$	$\widehat{=}$	finite ; f
$\Box f$	$\widehat{=}$	$\neg \Diamond \neg f$
fin f	$\widehat{=}$	$\Box$(empty $\supset$ f)
$\Diamond_i f$	$\widehat{=}$	f ; true
$\Box_i f$	$\widehat{=}$	$\neg \Diamond_i \neg f$
$w \; ? \; f : g$	$\widehat{=}$	$(w \wedge f) \vee (\neg w \wedge g)$