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

- 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?
Interval Temporal Logic
Syntax

**Expressions**

\[ e ::= \mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \Diamond v \mid \text{fin } v \]

**Formulae**

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

- \( \mu \) 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
Interval Temporal Logic

Syntax

Expressions

\[ e ::= \mu | a | A | g(e_1, \ldots, e_n) | \bigcirc v | \text{fin } v \]

Formulae

\[ f ::= p(e_1, \ldots, e_n) | \neg f | f_1 \land f_2 | \forall v \cdot f | \text{skip} | f_1 ; f_2 | f^* \]

\( \mu \) 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
Interval Temporal Logic
Informal Semantics

State Formula

Skip (Unit Interval)

Chop (Sequence)

Chopstar (Iteration)
Policy Rule

Expresses individual protection requirements in the form:

\[ \text{premise} \rightarrow \text{consequence} \]

- **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:

\[ f \rightarrow w \equiv \Box (\Diamond f \supset \text{fin } w) \]

where \( f \) stands for any ITL formula, and \( w \) is a state formula.
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:

$$E_{autho} \equiv \text{keep } (\circ done(s, o, a) \supset autho(s, o, a))$$
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.
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 \( C_{autho} \).
We use *temporal projection* to map between the more coarse policy reference interval and the fine grained RM specification.
Subject \( s \) is authorised to perform \( a \) on \( o \) if \( s \) was not acting in the role \textit{admin} in the state before.

\[
1 : \neg \text{in}(s, \text{admin}) \leftrightarrow \text{autho}(s, o, a)
\]

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.

\[
\text{enf}_{\text{pre}} \overset{\Delta}{=} \forall s \in S \cdot \\
H_{\text{in}, s, \text{admin}}[1], H_{\text{in}, s, \text{admin}}[0] \leftarrow H_{\text{in}, s, \text{admin}}[0], \text{in}(s, \text{admin})
\]

where \( H \) is a list of history variables for the observed subscript.
The (parallel) temporal assignment can be refined into the following sequence:

\[ \text{enf}_{pre} \equiv \text{for } s \text{ in } S : \{ \]
\[ H_{\text{in},s,\text{admin}}[1] := H_{\text{in},s,\text{admin}}[0]; \]
\[ H_{\text{in},s,\text{admin}}[0] := \text{in}(s, \text{admin}) \]
\[ \} \]

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

\[ C_{\text{autho}} \equiv T \geq 1 \land \neg H_{\text{in},s,\text{admin}}[1] \]
Summary

- 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

\[
\begin{align*}
\bigcirc f & \equiv \text{skip} ; f \\
\text{more} & \equiv \text{skip} ; \text{true} \\
\text{empty} & \equiv \neg \text{more} \\
\text{inf} & \equiv \text{true} ; \text{false} \\
\text{finite} & \equiv \neg \text{inf} \\
\lozenge f & \equiv \text{finite} ; f \\
\lozenge f & \equiv f ; \text{true} \\
\square f & \equiv \neg \lozenge \neg f \\
\text{fin } f & \equiv \square (\text{empty } \supset f) \\
\lozenge f & \equiv (w \land f) \lor (\neg w \land g)
\end{align*}
\]