A (Co)inductive System Calculus for Security Properties

[New title suggestions are welcome!]

Eric Rothstein Morris
Supervisor: Joachim Posegga

Chair of IT Security
University of Passau
er@sec.uni-passau.de

ESORICS 2015 - PhD Symposium
October 2, 2015

Enforcement

Let Sys be a set of systems.
Let $\mathrm{P}\colon \mathit{Sys} \to \{\,\mathrm{FALSE}, \mathrm{TRUE}\,\}$ be a system property.

Definition

A **sound enforcer** of P is a mechanism $\mathit{enf}_{\mathrm{P}}\colon \mathit{Sys} \to \mathit{Sys}$ such that, for all $\sigma \in \mathit{Sys}$, $\mathit{enf}_{\mathrm{P}}(\sigma)$ satisfies P.

Definition

An enforcer $\mathit{enf}_{\mathrm{P}}$ is **transparent** if and only if whenever σ satisfies P, then $\mathit{enf}_{\mathrm{P}}(\sigma) = \sigma$.

Enforcement

Relevant questions:

- What is $\mathcal{Sys}$?
- Sound and transparent enforcer for all properties?

Usually:

- Systems: C, JavaScript, automata, hardware, etc.
- Properties: not vulnerable to ν, confidentiality, etc.

Know the power of your enforcer

Enforcing via Equations: An Artificial Toy Example

Consider the following

- Let $\mathcal{S}ys = \mathbb{R} \to \mathbb{R}$
- Let $\mathrm{P} : \mathcal{S}ys \to \{ \text{FALSE}, \text{TRUE} \}$ defined, for $f \in \mathcal{S}ys$, by

$$\mathrm{P}(f) = f(r) \geq 0, \quad \text{for all } r \in \mathbb{R}.$$

- Let $|\cdot|: \mathcal{S}ys \to \mathcal{S}ys$ defined, for $f \in \mathcal{S}ys$, by

$$|f|(r) = \begin{cases} f(r), & \text{if } f(r) \geq 0; \\ -f(r), & \text{otherwise;} \end{cases}$$

The function $|\cdot|$ is one sound and transparent enforcer for P

BIOMICS

Enforcing via Equations: An Artificial Toy Example

Your competition proposes

$$\mathit{enf}_{\mathrm{P}}(f)(r) = \begin{cases} f(r), & \text{if } f(r) \geq 0; \\ 0, & \text{otherwise}; \end{cases}$$

- Enforcement policy: use $\mathit{enf}_{\mathrm{P}}$ or $|\cdot|$?

Enforcement: not only about what, but also about how.

Verifying vs. enforcing

- Verify: prove $f(r) \geq 0$ for all $r \in \mathbb{R}$ (maybe hard).
- Enforce: use $|f|$ or $\mathit{enf}_{\mathrm{P}}$ instead of f (easy)

Motivation

It would be nice if we could do the same for complex systems and for practical security properties

Can we actually do this?

Motivation

It would be nice if we could do the same for complex systems and for practical security properties

Can we actually do this?

Hopefully yes, using coinductive calculus

Before we continue

I will try to convince you that...

- Coinduction: break systems apart, rebuild them back.
- Enforcement: rebuild systems so they satisfy a property.
- Implementation: equations lazily evaluated in Haskell.

Coinduction: Breaking Streams Appart

Streams (Single-threaded, non-interactive systems)

- Let $\mathbb{R}^\omega = \{ [r_0, r_1, \ldots] \mid r_i \in \mathbb{R} \}$
- Let $\texttt{head}\colon \mathbb{R}^\omega \to \mathbb{R}$ defined by

$$\texttt{head}([r_0, r_1, \ldots]) = r_0.$$

- Let $\texttt{tail}\colon \mathbb{R}^\omega \to \mathbb{R}^\omega$ defined by

$$\texttt{tail}([r_0, r_1, \ldots]) = [r_1, \ldots].$$

Stream σ is coinductively defined by its `head` and `tail`

Coinduction: Rebuilding Streams from Pieces

Let $\texttt{pack}\colon \mathbb{R} \times \mathbb{R}^\omega \to \mathbb{R}^\omega$ **be defined by**

$$\texttt{pack}(r, [r_0, r_1, ...]) = [r, r_0, r_1, ...]$$

$\texttt{pack}$ **is the "compiler" of the specification** $\langle r, [r_0, r_1, ...]\rangle$

$$\mathbb{R} \times \mathbb{R}^\omega \cong \mathbb{R}^\omega$$

Modify the $\texttt{head}$ **and/or** $\texttt{tail}$ **to obtain a different stream.**

$$\mathit{enf}_{\mathrm{P}}(\sigma) = \texttt{pack}(f \circ \texttt{head}(\sigma), g \circ \texttt{tail}(\sigma))$$

Another Toy Example

Define enforcers using $\texttt{head}$**,** $\texttt{tail}$ **and** $\texttt{pack}$

Let $|\cdot|\colon \mathbb{R}^\omega \to \mathbb{R}^\omega$ defined, for $\sigma \in \mathbb{R}^\omega$ by

$$|\sigma| = \begin{cases} \texttt{pack}(\langle \texttt{head}(\sigma), |\texttt{tail}(\sigma)|\rangle), & \text{if } \texttt{head}(\sigma) \geq 0; \\ |\texttt{tail}(\sigma)|, & \text{otherwise;} \end{cases} \tag{1}$$

$|\cdot|$ **soundly and transparently enforces "always** ≥ 0**"**

Equation (1) **is a behavioural (differential) equation.**

BIOMICS

From Streams to Arbitrary Types

Let X be a Haskell type implementing:

- `observe`: $X \to \mathbb{R}$
- `next`: $X \to X$

Enforce "always ≥ 0" on X using $|\cdot|$ by projecting X into $\mathbb{R}^\omega$

From Streams to Arbitrary Types

Let X be a Haskell type implementing:

- $\texttt{observe}\colon X \to \mathbb{R}$
- $\texttt{next}\colon X \to X$

Enforce "always ≥ 0" on X using $|\cdot|$ by projecting X into $\mathbb{R}^\omega$

Let $[\![\cdot]\!]\colon X \to \mathbb{R}^\omega$ be defined, for $x \in X$, by

$$[\![x]\!] = \texttt{pack}(\langle \texttt{observe}(x), [\![\texttt{next}(x)]\!]\rangle)$$

$|[\![x]\!]|$ satisfies "always ≥ 0" and x and $[\![x]\!]$ are behaviourally equivalent.

BIOMICS

Non-interference

Let I be a set of inputs, $\texttt{lvl}\colon I \to \{\,\mathcal{L}, \mathcal{H}\,\}$ be an input classification function, and X be a Haskell type implementing:

- $\texttt{observe}\colon X \to I \to \mathbb{R}$ (an $\mathcal{L}$-channel)
- $\texttt{next}\colon X \to I \to X$

Non-interference: the presence of $\mathcal{H}$-actions does not impact $\mathcal{L}$-channels.

$$\texttt{observe}(\mathit{enf}_{\mathrm{P}}(\sigma), i) = \begin{cases} \texttt{observe}(\sigma, i), & \text{if } \texttt{lvl}(i) = \mathcal{L}; \\ \varepsilon, & \text{otherwise.} \end{cases}$$

$$\texttt{next}(\mathit{enf}_{\mathrm{P}}(\sigma), i) = \begin{cases} \mathit{enf}_{\mathrm{P}} \circ \texttt{next}(\sigma, i), & \text{if } \texttt{lvl}(i) = \mathcal{L}; \\ \mathit{enf}_{\mathrm{P}}(\sigma), & \text{otherwise.} \end{cases}$$

Contribution

Illustrate how systems, properties and enforcement mechanisms can be brought down to the same abstraction level; resulting in a practical framework for the enforcement of security properties.

BIOMICS

Objective

Find and solve systems of behavioural equations to obtain systems that satisfy security properties.

Milestones:

- Find equations that define security properties
  - Prove expressivity: "benchmark" properties
  - Classify properties according to enforceability
- Develop tool support: Haskell

Questions

Questions?

Thank you for your attention!