An Approach to Modular and Testable Security Models of Real-world Health-care Applications

Achim D. Brucker[1] Lukas Brügger[2]
Paul Kearney[3] Burkhart Wolff[4]

[1]SAP Research, Germany

[2]Information Security, ETH Zürich, Switzerland

[3]Security Futures Practice, BT Innovate & Design, UK

[4]Université Paris-Sud, France

SACMAT 2011

Motivation

- **Observation:**
  Health care systems
  - are large (e.g. nationwide, central databases)
  - are complex from a system's perspective
  - handle sensitive data
  - require complex security and access control policies

Motivation

- **Observation:**
  Health care systems
  - are large (e.g. nationwide, central databases)
  - are complex from a system's perspective
  - handle sensitive data
  - require complex security and access control policies
- **Question:**
  Can we implement health-care systems correctly (safe and securely)?

Motivation

- **Observation:**
  Health care systems
  - are large (e.g. nationwide, central databases)
  - are complex from a system's perspective
  - handle sensitive data
  - require complex security and access control policies
- **Question:**
  Can we implement health-care systems correctly (safe and securely)?
- **Contribution:**
  - An approach for model-based compliance testing of access control policies of large, complex systems,
  - instantiated with the AC framework of the NPfIT (National Programme for IT in the NHS in England).

Access Control Framework of the NPfIT

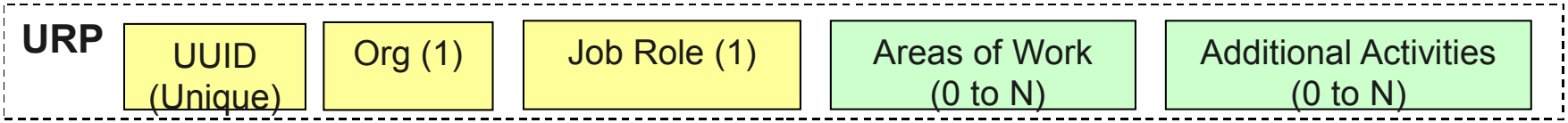
- The NPfIT consists of
  - a couple of central services, e.g. care records of every patient held on the Spine
  - a large number of applications deployed locally
- Different concepts govern which users may access which functions and which health care data, according to
  - role of the user in the NHS: RBAC
  - his relationship to the patient: Legitimate Relationships
  - consent status from the patient: Patient's Consent
  - if anyone has "sealed" any data: Sealed Envelopes

RBAC (NPfIT Version)

Guiding Principle

A user is allowed to access only those functions that he needs to perform his job duties.

- *Activities* (ca. 150) describe generic functionality in a hierarchy
- Each application defines
  - its set of application *functions*
  - maps the functions to activities
- Each user is given one or more **URP** (User Role Profile)
- The URP determines which activities you're allowed to perform
- A national database (NRD) defines the mapping from roles and areas of work to activities



Legitimate Relationships (LR)

Guiding Principle

A user is only allowed to access the data of patients in whose care he is actually involved in.

- Users are assigned to hierarchically ordered workgroups, reflecting the organisational structure of a workplace.
- A legitimate relationship is a relationship between a workgroup or a user and a patient.
- There are ten types of LRs.
- LRs are dynamic and are attributed and removed as part of the application's workflow.

Example

- User **Alice**, in her role as a *Clerical*,
  is a member of the workgroup *orthopaedics*
- Patient **Pablo** has an existing LR with *orthopaedics*
- **Pablo** is being referred, and now treated by the workgroup *surgery*
- *Nurse* **Bob** from *surgery* wants to read the care record of **Pablo**

Example

- User **Alice**, in her role as a *Clerical*,
  is a member of the workgroup *orthopaedics*
- Patient **Pablo** has an existing LR with *orthopaedics*
- **Pablo** is being referred, and now treated by the workgroup *surgery*
- *Nurse* **Bob** from *surgery* wants to read the care record of **Pablo**

Operation	RBAC	LR	Decision
readSCR Bob Pablo	$\surd$	x	Deny

Example

- User **Alice**, in her role as a *Clerical*, is a member of the workgroup *orthopaedics*
- Patient **Pablo** has an existing LR with *orthopaedics*
- **Pablo** is being referred, and now treated by the workgroup *surgery*
- *Nurse* **Bob** from *surgery* wants to read the care record of **Pablo**

Operation	RBAC	LR	Decision
readSCR Bob Pablo	√	x	Deny
addLR Alice Pablo Surgery Referral	√	√	Allow

Example

- User **Alice**, in her role as a *Clerical*, is a member of the workgroup *orthopaedics*
- Patient **Pablo** has an existing LR with *orthopaedics*
- **Pablo** is being referred, and now treated by the workgroup *surgery*
- *Nurse* **Bob** from *surgery* wants to read the care record of **Pablo**

Operation	RBAC	LR	Decision
readSCR Bob Pablo	√	x	Deny
addLR Alice Pablo Surgery Referral	√	√	Allow
readSCR Bob Pablo	√	√	Allow

Access Control Policy of the NPfIT

- Observations:
  - AC principles of the NPfIT are described in a set of informal documents
  - Hard to grasp the core concepts
  - No central enforcement
  - Applications have to conform to different versions of these principles
  - Lack of tools for ensuring the quality of the implementation (e.g. testing)
  - The policies are large and complex

Access Control Policy of the NPfIT

- Observations:
  - AC principles of the NPfIT are described in a set of informal documents
  - Hard to grasp the core concepts
  - No central enforcement
  - Applications have to conform to different versions of these principles
  - Lack of tools for ensuring the quality of the implementation (e.g. testing)
  - The policies are large and complex
- Our Approach for formalisation of such policies:
  - Identify the core concepts of access control models
  - Provide a formal framework (UPF) that covers many AC models (in Isabelle/HOL)
  - Instantiate the UPF with the NPfIT policies

Unified Policy Framework (UPF)

- Key Idea: A policy as a policy decision function.

Unified Policy Framework (UPF)

- Key Idea: A policy as a policy decision function.

  type_synonym (α,β) policy $= \alpha \rightharpoonup \beta$ decision $(* \ \alpha \mapsto \beta \ *)$

- The enumeration type decision is defined as:

  datatype β decision $=$ allow β $\mid$ deny β

- Types:
  - α: input data (e.g. arguments, system state, security context, network packets)
  - β: output data (e.g. system state, security contexts, return messages, network packets, nothing)

- An important case are transition policies:

$$\iota \times \sigma \mapsto \mathsf{o} \times \sigma$$

  and higher-order policies (policies transforming policies).

UPF: Policy Constructors

The UPF provides a standard set of constructors and combinators to specify policies in functional-programming style:

- Update:

  $p(x \mapsto t)$

- Elementary policies are

  $\emptyset$

  $\forall D_f \qquad (DenyAll)$

- The operation *override* $\_ \bigoplus \_$ combines small policies (*rules*) to more complex ones in a first-fit manner, e.g.:

  $p_1 \bigoplus \cdots \bigoplus p_n \bigoplus \forall D_f$

- Parallel composition of policies:

  definition $\_ \bigotimes_{\vee D} \_ :: (\alpha \mapsto \beta) \Rightarrow (\gamma \mapsto \delta) \Rightarrow (\alpha \times \gamma \mapsto \beta \times \delta)$

NPfIT Concepts and Operations I

- Formalisation of the main concepts:
  - Care Records
  - the Database (*Spine*)
  - User Role Profiles,
  - etc.
- Model the operations: abstract descriptions of the policy-related system behaviours.
- Example:

[readSCR Bob Pablo,
addLR Alice Pablo Surgery Referral ,
readSCR Bob Pablo]

NPfIT Concepts and Operations II

- Formalisation of the state:
  - Care Records Database (Spine)
  - Security Context Σ (LRs and Workgroups)
  - User Context υ (allocated URPs)
- Possible outputs of the system (as datatype)

Modelling Strategy (Generic)

- Formalise the individual rules, e.g.:

  LR_Rule1 ((editEntry u p e_i e), Σ) = (if hasLR u p Σ
  then Some (allow ()) else Some (deny ()))

- For each of the four AC concepts, combine the rules (using the override operator with a default DenyAll)

Modelling Strategy (Generic)

- Formalise the individual rules, e.g.:

  LR_Rule1 ((editEntry u p e_i e), Σ) = (if hasLR u p Σ
  then Some (allow ()) else Some (deny ()))

- For each of the four AC concepts, combine the rules (using the override operator with a default DenyAll)
- Make a parallel composition of the four policies (where a Deny wins)

definition appPolicy :: (Operation $\times$ Spine $\times \Sigma \times \upsilon$) $\mapsto$ unit
where appPolicy = C_1 o$_f$ ((C_1 o$_f$ ((C_1 o$_f$
(PCPolicy
$\bigotimes_{\vee D}$ SEPolicy) o C_2)
$\bigotimes_{\vee D}$ LRPolicy) o C_3)
$\bigotimes_{\vee D}$ AppRBACPolicy) o C_4

Modelling Strategy (Generic)

- Formalise the individual rules, e.g.:

  LR_Rule1 ((editEntry u p e_i e), Σ) = (if hasLR u p Σ
  then Some (allow ()) else Some (deny ()))

- For each of the four AC concepts, combine the rules (using the override operator with a default DenyAll)
- Make a parallel composition of the four policies (where a Deny wins)

  definition appPolicy :: (Operation $\times$ Spine $\times \Sigma \times \upsilon$) $\mapsto$ unit
  where appPolicy = C_1 o$_f$ ((C_1 o$_f$ ((C_1 o$_f$
  (PCPolicy
  $\bigotimes_{\vee D}$ SEPolicy) o C_2)
  $\bigotimes_{\vee D}$ LRPolicy) o C_3)
  $\bigotimes_{\vee D}$ AppRBACPolicy) o C_4

- Model the state transitions for each part, combine them, and combine the policy with the state transition to a transition policy:

  (Operation $\times$ Spine $\times \Sigma \times \upsilon$) $\mapsto$(Output $\times$ Spine $\times \Sigma \times \upsilon$)

Conformance Testing of Access Control Policies

Problem:

How can we ensure that a real system complies to the modelled security policy?

- Using a formal model of these principles, we can generate test data (an operation or a sequence of operations together with the expected output)
- The test sequences usually satisfy some interesting properties (*Nurses are not allowed to access records of patients to which they do not have an LR*)
- According to such test data, test scripts can be generated to perform the tests automatically

Test Data Generation using HOL-TestGen

- Standard test specification:

  $\mathsf{P} \wedge (\ \mathsf{TransitionPolicy}\ \ (\sigma 0,\ \mathsf{is}\,) = \mathsf{os}) \Longrightarrow$
  $\mathsf{PUT}\ (\sigma 0,\ \mathsf{is}\,) = \mathsf{os}$

- Essential: Specification of P (predicate on the input sequence *is* and the output sequence *os*)

Test Data Generation using HOL-TestGen

- Standard test specification:

  $P \wedge (\text{TransitionPolicy}\ (\sigma 0,\ \text{is}) = \text{os}) \Longrightarrow$
  $\text{PUT}\ (\sigma 0,\ \text{is}) = \text{os}$

- Essential: Specification of P (predicate on the input sequence *is* and the output sequence *os*)
- Returning test data:

  $\text{PUT}\ (\sigma 0, [\text{readSCR Bob Pablo, addLR Alice Pablo Surgery Referral},$
  $\text{readSCR Bob Pablo}]) =$
  $[\text{deny}\ (),\ \text{allow}\ (),\ \text{allow SCR\_pablo}]$

Test Data Generation using HOL-TestGen

- Standard test specification:

  $P \wedge (\text{TransitionPolicy}\ (\sigma 0,\ \text{is}) = \text{os}) \Longrightarrow$
  $\text{PUT}\ (\sigma 0,\ \text{is}) = \text{os}$

- Essential: Specification of P (predicate on the input sequence *is* and the output sequence *os*)
- Returning test data:

  $\text{PUT}\ (\sigma 0, [\text{readSCR Bob Pablo, addLR Alice Pablo Surgery Referral},$
  $\text{readSCR Bob Pablo}]) =$
  $[\text{deny}\ (),\ \text{allow}\ (),\ \text{allow SCR\_pablo}]$

- Such sequences can be used to automatically test a real implementation of an application of the NPfIT, e.g. based on web services (currently done on mock-up applications with WSDL support)

Conclusion

- Results:
  - Generic framework for formalising access control policies,
  - and for performing model-based conformance testing.
  - Shown to be able to handle large, complex, real-world systems by instantiating it with the AC framework of the NPfIT.
  - Techniques and formalisations can be reused in very different scenarios.
- Observation:
  any framework for real-world policy formalisations must be
  - flexible,
  - modular
- Future Work:
  - Framework should be accessible to engineers
  - Advanced universal techniques for normalisation, equivalence-checking, refinement, and test-generation of policies
  - Executing the tests on a real NPfIT application