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

Achim D. Brucker\textsuperscript{1} \hspace{1em} Lukas Brügger\textsuperscript{2}  
Paul Kearney\textsuperscript{3} \hspace{1em} Burkhart Wolff\textsuperscript{4}

\textsuperscript{1}SAP Research, Germany  
\textsuperscript{2}Information Security, ETH Zürich, Switzerland  
\textsuperscript{3}Security Futures Practice, BT Innovate & Design, UK  
\textsuperscript{4}Université Paris-Sud, France

SACMAT 2011
1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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).
Motivation

Access Control in the NPfIT

The Unified Policy Framework

Modelling NPfIT Policies in UPF

Testing NPfIT Applications

Conclusion
1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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**

<table>
<thead>
<tr>
<th>Operation</th>
<th>RBAC</th>
<th>LR</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>readSCR Bob Pablo</td>
<td>✓</td>
<td>x</td>
<td>Deny</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>RBAC</th>
<th>LR</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>readSCR Bob Pablo</td>
<td>✓</td>
<td>✗</td>
<td>Deny</td>
</tr>
<tr>
<td>addLR Alice Pablo Surgery Referral</td>
<td>✓</td>
<td>✓</td>
<td>Allow</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Operation</th>
<th>RBAC</th>
<th>LR</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>readSCR Bob Pablo</td>
<td>✓</td>
<td>✗</td>
<td>Deny</td>
</tr>
<tr>
<td>addLR Alice Pablo Surgery Referral</td>
<td>✓</td>
<td>✓</td>
<td>Allow</td>
</tr>
<tr>
<td>readSCR Bob Pablo</td>
<td>✓</td>
<td>✓</td>
<td>Allow</td>
</tr>
</tbody>
</table>
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
The Unified Policy Framework

1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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.
  
  \[
  \text{type_synonym} \ (\alpha, \beta) \ \text{policy} = \alpha \rightarrow \beta \ \text{decision} \ (\ast \ \alpha \mapsto \beta \ \ast)
  \]

- The enumeration type decision is defined as:
  
  \[
  \text{datatype} \ \beta \ \text{decision} = \text{allow} \ \beta \ \mid \text{deny} \ \beta
  \]

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

- An important case are transition policies:
  
  \[
  \iota \times \sigma \mapsto \sigma \times \sigma
  \]

  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**
  \[ \emptyset \]
  \[ \forall D_f \quad (DenyAll) \]

- The operation *override* combines small policies (*rules*) to more complex ones in a first-fit manner, e.g.:
  \[ p_1 \oplus \cdots \oplus p_n \oplus \forall D_f \]

- **Parallel composition of policies:**
  \[ \text{definition} \quad \bigotimes \lor_D \colon (\alpha \mapsto \beta) \Rightarrow (\gamma \mapsto \delta) \Rightarrow (\alpha \times \gamma \mapsto \beta \times \delta) \]
1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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 $\Sigma$ (LRs and Workgroups)
  - User Context $\nu$ (allocated URPs)

- Possible **outputs** of the system (as datatype)
Modelling Strategy (Generic)

- Formalise the individual rules, e.g.:
  \[ LR\_Rule1 ((\text{editEntry } u \ p \ e_i \ e), \ \Sigma) = (\text{if hasLR } u \ p \ \Sigma \ \\
  \text{then Some (allow ()) else Some (deny ())}) \]

- 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.:
  \[
  \text{LR\_Rule1} \left( \text{editEntry } u \ p \ e \ i \ e, \ \Sigma \right) = \text{if hasLR } u \ p \ \Sigma \text{ then Some }(\text{allow }()) \text{ else Some }(\text{deny }())
  \]

- 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)

\begin{verbatim}
definition appPolicy :: (Operation × Spine × Σ × υ) ↦→ unit
where appPolicy = C₁ oₖ ((C₁ oₖ ((C₁ oₖ (PCPolicy
        ⊗₆ SEPolicy) o C₂ )
        ⊗₆ LRPolicy) o C₃ )
        ⊗₆ AppRBACPolicy) o C₄
\end{verbatim}
Modelling Strategy (Generic)

- Formalise the individual rules, e.g.:
  \[ LR_{-Rule1} \left( \left( \text{editEntry } u \ p \ e_i \ e \right), \ \Sigma \right) = (\text{if hasLR } u \ p \ \Sigma \ \text{then Some (allow ()}) \ \text{else Some (deny ())}) \]

- 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)

\[ \text{definition appPolicy :: } (\text{Operation } \times \text{Spine } \times \Sigma \times \nu) \mapsto \text{unit} \]
\[ \text{where appPolicy } = \text{C}_1 \circ (\text{C}_1 \circ (\text{C}_1 \circ (\text{PCPolicy} \ \otimes \ \vee_{\text{D}} \ \text{SEPolicy}) \ \circ \ \text{C}_2 ) \)
\[ \otimes \ \vee_{\text{D}} \ \text{LRPolicy}) \ \circ \ \text{C}_3 ) \)
\[ \otimes \ \vee_{\text{D}} \ \text{AppRBACPolicy}) \ \circ \ \text{C}_4 \]

- Model the state transitions for each part, combine them, and combine the policy with the state transition to a transition policy:
  \[ (\text{Operation } \times \text{Spine } \times \Sigma \times \nu) \mapsto (\text{Output } \times \text{Spine } \times \Sigma \times \nu) \]
1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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
Testing NPfIT Applications

Test Data Generation using HOL-TestGen

- Standard test specification:
  \[ P \land (\text{TransitionPolicy}(\sigma_0, \text{is}) = \text{os}) \Rightarrow \text{PUT}(\sigma_0, \text{is}) = \text{os} \]

- **Essential**: Specification of \( P \) (predicate on the input sequence \( \text{is} \) and the output sequence \( \text{os} \))
Testing NPfIT Applications

Test Data Generation using HOL-TestGen

- Standard test specification:
  \[ P \land (\text{TransitionPolicy}(\sigma_0, is) = os) \implies \text{PUT}(\sigma_0, is) = os \]

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

- Returning test data:
  \[ \text{PUT}(\sigma_0, [\text{readSCR Bob Pablo, addLR Alice Pablo Surgery Referral, readSCR Bob Pablo}]) = [\text{deny}(), \text{allow}(), \text{allow SCR}_{-\text{pablo}}] \]
Test Data Generation using HOL-TestGen

- **Standard test specification:**
  \[ P \land (\text{TransitionPolicy}(\sigma_0, \text{is}) = \text{os}) \implies \text{PUT}(\sigma_0, \text{is}) = \text{os} \]

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

- Returning test data:
  \[
  \text{PUT}(\sigma_0,[\text{readSCR Bob Pablo}, \text{addLR Alice Pablo Surgery Referral}, \text{readSCR Bob Pablo}]) = [\text{deny ()}, \text{allow ()}, \text{allow SCR_pablo}]
  \]

- 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)
1 Motivation

2 Access Control in the NPfIT

3 The Unified Policy Framework

4 Modelling NPfIT Policies in UPF

5 Testing NPfIT Applications

6 Conclusion
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