Language-specific vs. language-independent approaches: embedding semantics on a metamodel for testing and verifying access control policies

Yves Le Traon
Tejeddine Mouelhi
Franck Fleurey
Benoit Baudry
Context and motivations

Quality of a Metamodel for verification and testing purpose

- MDE approach vs. language specific one
- The today issue: Metamodel loses information (and semantics)
  - Loss: quality/abstraction?
  - Compared with language-specific techniques
- Problem tackled in the context of (MODELS’08)
  - Access control policies (RBAC, OrBAC, MAC, DAC, …)
  - Need to validate security policies
  - Technology independent qualification process
Outline

Using MDE for security
Access control meta-model
Verification checks of the model
Mutation testing for security
Case studies and results
Summary and conclusions
Language-specific vs. language-independent approaches

Metamodelling ???

MetaModel

Model

Instance

MMFruit

Semantics

Is-not-rotten
check-colour
cut

MBlueBerry

MApple

MBanana

MOrange

MStrawberry

Is-not-rotten
cut

Is-not-rotten
cut

Is-not-rotten
cut

Is-not-rotten
cut

Is-not-rotten
cut
Metamodelling

Language-specific vs. language-independent approaches

MetaModel

Model

Instance

MChicken

MRabbit

MLamb

MPig

MCow

MMFruit

Semantics

Is-not-rotten

seems-better(a, b)

cut

check-colour
Verifying and Testing Access Control Policies

MetaModel

Model

Instance

Language-specific vs. language-independent approaches

MMSecurityPolicy

Semantics
Is-conform() V&V
No-redundancies() Mutate()

MMSecurityPolicy

RBAC

MOrBAC

MDAC

Is-conform() Mutate()
Is-conform() Mutate()
Is-conform() Mutate()

R1 -> UserRole( romain Student )
R2 -> UserRole( yves Director )
R3 -> UserRole( alice Secretary )
R4 -> RolePermission( Student BorrowBook WorkingDays )
R5 -> RolePermission( Personnel ModifyAcnt WorkingDays )
R6 -> RolePermission( Director CreateAccount AllTime )

POLICY systemDAC (DAC)
R1 -> DACRule(Tim r file1)
R2 -> DACRule(Tim x file1)
R3 -> DACRule(Admin cp file1)
R4 -> DACRule(Admin r file1)
R5 -> DACRule(Admin w file1)
R6 -> DACRule(Admin x file1)
Overview of the approach (1)

- Model security early phases
  - Better capture the security concern
  - Allow for validation
  - Use DSLs accessible to domain experts
  - Independent of the security platform
- Few assumptions on the functional code
Overview of the approach (2)

- Generate platform specific security policy
  - Using model transformation / code generation
Overview of the approach (3)

- Provide operational mapping
Overview of the approach (4)

- Compose security code with functional code
Overview of the approach (5)

Language-specific vs. language-independent approaches

- Requirements
  - Platform independent security model
    - Platform specific security code (PDP)
      - Weaving
        - Operational Mapping (PEP)
          - Functional code
            - Running Code
              - Security Tests
Overview of the approach (5)

Language-independent verification checks

Requirements

Platform independent security model

Language-independent mutations

Operational Mapping (PEP)

Platform specific security code (PDP)

Weaving

Functional code

Running Code

Security Tests
Outline

Using MDE for security

Access control meta-model
Verification checks of the model
Mutation testing for security
Case studies and results
Summary and conclusions
Modeling access control policies

- Most formalisms are rule based
  - Different types of rules
  - Different types of parameters

- Examples
  - OrBAC
    - R1 - Permission (Library Student Borrow Book WorkingDays)
    - R2 - Prohibition (Library Student Borrow Book Holidays)
  - RBAC
    - R3 – UserRole (Alice Student)
    - R4 – RolePermission (Student BorrowBook WorkingDays)

Need to model both rules and rule types
Modeling rule types (1)

- Define elements types
- Define rules types
Modeling rules (1)

- Model rules and their parameters
Modeling rules (1)

- Model rules and their parameters

![Diagram showing the relationship between PolicyType, RuleType, Policy, Parameter, and Rule]
Outline

Using MDE for security
Access control meta-model
Verification checks of the model
Mutation testing for security
Case studies and results
Summary and conclusions
Simple verification checks

- `no_conflicts()`: checks the absence of conflicts. It essentially involves checking that there are no rules having the same parameters and having different types.

- `no_redundancies()`: checks that the security policy is minimal, which means that no rule appears more than once.
Verification checks

Some checks are not relevant for all access control languages:

no_conflict

MetaModel

HRBAC

Is-conform()

Mutate()

MOrBAC

Is-conform()

Mutate()

MDAC

Is-conform()

Mutate()

Model

Semantics

Is-conform()  V&V

No-redundancies()

Mutate()

Instance

R1 -> UserRole( roain Student )
R2 -> UserRole( yves Director )
R3 -> UserRole( alice Secretary )
R4 -> RolePermission( Student BorrowBook WorkingDays )
R5 -> RolePermission( Personnel ModifyAcnt WorkingDays)
R6 -> RolePermission( Director CreateAccount AllTime )
Advantages and limitations

- It works
  - Same verification for any rule-based access control languages

- But
  - The verification checks
    - Are very rough
    - Need complementary specific checks
  - Not specific to access control
    - rule-based system
    - Too generic?

Disappointing?
Outline

Using MDE for security
Access control meta-model
Verification checks of the model
**Mutation testing for security**
Case studies and results
Summary and conclusions
Mutation analysis: certifying tests with a common framework

- **Objective**
  - Qualify tests with respect to security

- **Principle**
  - Inject faults in the security model
  - Check that the tests can catch faults

- **Defining faults models at a generic level**

- **Defining the analysis at the generic level**
  - The analysis can be performed on any policy defined according to any policy model
Mutation operators

- 5 Generic operators
  - ANR → new rule
  - RER → Remove existing rule
  - PPR → Replace rule parameter
  - RTT → Modify rule type
  - PPD → A parameter replaced by a descendant
Operators in the Metamodel
**Removes an existing rule**

class RER inherits SPMutator {
    method mutate(p : Policy) : set Policy[*] is do
        var mutant : Policy
        result := Set<Policy>.new
        p.rules.each{ r |
            // create mutated policy
            mutant := p.copy
            mutant.name := p.name + "-RER-" + r.name
            mutant.rules.remove(mutant.rules.detect{x | x.name == r.name})
            result.add(mutant)
        }
        end
    }
}
Outline

Using MDE for security
Access control meta-model
Verification checks of the model
Mutation testing for security

Case studies and results
Summary and conclusions
Empirical validation

• Two questions
  – Is the approach feasible / practical ?
  – Are the generic mutation operator meaningful ?

• Three case studies

<table>
<thead>
<tr>
<th></th>
<th># classes</th>
<th># methods</th>
<th>LOC (executable statements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>62</td>
<td>335</td>
<td>3204</td>
</tr>
<tr>
<td>VMS</td>
<td>134</td>
<td>581</td>
<td>6077</td>
</tr>
<tr>
<td>ASMS</td>
<td>122</td>
<td>797</td>
<td>10703</td>
</tr>
</tbody>
</table>
Results

- Generic mutation operator implementation
- Generate more mutants
- Includes all specific mutants

<table>
<thead>
<tr>
<th>System</th>
<th>generic mutants</th>
<th>specific mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>1044</td>
<td>371</td>
</tr>
<tr>
<td>VMS</td>
<td>1572</td>
<td>1426</td>
</tr>
<tr>
<td>ASMS</td>
<td>3088</td>
<td>2056</td>
</tr>
</tbody>
</table>
### Mutation scores generic/specific

<table>
<thead>
<tr>
<th>Mutants</th>
<th>Basic Mutants (func. Tests)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
<td>LMS</td>
</tr>
<tr>
<td><strong>Generic mutants</strong></td>
<td>72%</td>
</tr>
<tr>
<td><strong>Specific mutants</strong></td>
<td>78%</td>
</tr>
<tr>
<td><strong>Delta</strong></td>
<td><strong>-6%</strong></td>
</tr>
</tbody>
</table>

### Robustness test cases

<table>
<thead>
<tr>
<th>Mutants</th>
<th>ANR mutants (sec. tests)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
<td>LMS</td>
</tr>
<tr>
<td><strong>Generic mutants</strong></td>
<td>13%</td>
</tr>
<tr>
<td><strong>Specific mutants</strong></td>
<td>17%</td>
</tr>
<tr>
<td><strong>Delta</strong></td>
<td><strong>-4%</strong></td>
</tr>
</tbody>
</table>
Using MDE for security
Access control meta-model
Generation of security components
Mutation testing for security
Case studies and results

Summary and conclusions
Summary and Conclusion: what is a ‘good meta-model’ for V&V?

- Generic meta-model for Access Control Policies
  - Platform independent (RBAC, HRBAC, OrBAC, MAC, DAC)

- Certification process defined at MM
  - verification
  - mutation testing

- Open questions
  - A specific “language-independent” metamodel
  - Metamodels Composition
Thank you!

Questions?
Overview of the approach

Requirements
- Using a P.I. security DSL
- Manageable by domain experts
- Separated from other requirements
- Supports consistency checks
- Supports security policy mutation

Platform independent Security Model
- Reusable transformations
- And code generators

Platform specific Security code (PDP)
- Ex. XACML

Operational Mapping (PEP)
- Specified by a developer
- Validated using mutation testing

Functional code
- Produced using regular Software Engineering techniques

Weaving
- Automated Weaving using AOP techniques

Running Code
- Produced from the requirements and security model
- Improved and validated through security mutation

Security Tests
- Validated using mutation testing

(1) (2) (3) (4) (5)
Implementation in EMF
Outline

Using MDE for security
Access control meta-model
**Generation of security components**
Mutation testing for security
Case studies and results
Summary and conclusions
Generation of security components

- Generate the Platform specific security code (PDP)
  - Process the policy model
  - Target existing technologies such as XACML
  - Typical MDE model to code

- Link the security code with the application (PEP)
  - No assumptions on the way the application is developed
  - Needs to be systematic
  - Needs to be automated
Rule R1 - Permission ( Library Student Borrow Book WorkingDays )

- <Rule RuleId="R1" Effect='Permit'>
  - <Target>
    - <Subjects>
      - <Subject>
        - <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">BORROWER</AttributeValue>
            <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  - <Resources>
    - <Resource>
      - <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">BOOK</AttributeValue>
            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  - <Actions>
    - <Action>
      - <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">BORROW</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
Weaving in the application

- Using AspectJ
- Example:

```java
pointcut borrowBookCall(User user, Book book):
    target(BookService) && call(void borrowBook(User, Book)) && args(user, book);

before(User user, Book book) throws SecurityPolicyViolationException: borrowBookPC(user, book) {
    // call security policy service to check for security rule
    Utils.checkSecurity(user, LibrarySecurityModel.BORROWBOOK_METHOD,
                        LibrarySecurityModel.BOOK_VIEW,
                        ContextManager.getTemporalContext());
}
```
Verification checks

- Some checks are not relevant for all access control languages

<table>
<thead>
<tr>
<th></th>
<th>Policy_is_conform</th>
<th>No_conflicts</th>
<th>No_redundancies</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBAC</td>
<td>y</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>HRBAC</td>
<td>y</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>OrBAC</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>DAC</td>
<td>y</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>MAC</td>
<td>y</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>
Modeling rule types (2)

- Example: OrBAC Model
Modeling rules (2)

- Example

R1 - Permission (Library Student Borrow Book WorkingDays)