Rigorous Analysis of UML Access Control Policy Models

Wuliang Sun, Robert France and Indrakshi Ray

Computer Science Department
Colorado State University
Fort Collins, Colorado
Security Policies

- Security policies
  - Expressed by different languages
    - XACML, OWL/RDF, CIM/SPL, PONDER, UML
- Errors in policy models
  - Can cause security breaches that have serious consequences
  - Correcting the errors once policies are deployed can be expensive
- Need error detection before policies are deployed
Motivation

- We use UML (Unified Modeling Language) for policy specification
  - UML together with OCL can be used to provide a formal and graphical representation of security policies
  - UML is the de facto specification language used in the software industry
  - UML policy models can be transformed to code using existing technologies
Motivation

- Using UML (Unified Modeling Language) as a policy specification language
  
  - Challenge: few mature tools supporting rigorous analysis of UML models
  
  - Solution: Transform UML to Alloy for automated model analysis
Approach

- Rigorous analysis of UML access control policy models
  - Front-end: use UML to describe security policies
  - Back-end: use Alloy Analyzer to analyze the modeled properties
  - Transformation between UML and Alloy: obviate the need for security designers to understand the Alloy language
Approach Overview
Background: SUDA

- Scenario-based UML Design Analysis (SUDA)
  - A design class model with operation specifications is transformed to a static model of behavior, called a snapshot model.
  - A snapshot is an object configuration that represents a state of the system at a particular time.
  - A snapshot transition describes the behavior of an operation in terms of its before and after effect on the system state.
Background: Snapshot Model

- Example of a snapshot model generated from a class model
Background: Snapshot Model

- Example of a snapshot model generated from a class model
Background: Snapshot Model

- Example of a snapshot model generated from a class model

UML Class Model

UML Snapshot Model
Background: Snapshot Model

- Example of a snapshot model generated from a class model

UML Class Model  
UML Snapshot Model
Background: Snapshot Model

- Example of a snapshot model generated from a class model
Background: SUDA

- SUDA vs. Our approach
  - SUDA: is the given scenario supported by the UML class model?
  - Our approach: is there a scenario supported by the UML class model that starts in a specified valid state and ends in a specified invalid state?
Approach Overview
Background: Dynamic Analysis using Alloy

- Alloy
  - Model = signatures + facts + predicates (operation specifications)
  - Facts: define constraints on the elements of the model
  - Predicates: probe model
  - Trace mechanism: specify a fact to associate the transitions triggered by operation invocations with states defined by signatures
Snapshot Model to Alloy Model Transformation

Step 1: Transform each class that is part of Snapshot class to a signature in Alloy

```
sig Role{}
sig User{}
```
Step 2: Transform the Snapshot class to a Snapshot signature containing fields that specify the object configurations within a snapshot

```alloy
sig Snapshot {
  // Object fields
  roles: set Role, users: set User,
  // Link fields
  ASSIGN: User set->set Role,
} {
  // Linked objects must exist in the snapshot
  ASSIGN=ASSIGN>:roles&users<: ASSIGN
}
```
Snapshot Model to Alloy Model Transformation

- Step 2: Transform the Snapshot class to a Snapshot signature containing fields that specify the object configurations within a snapshot.

```alloy
sig Snapshot{
  // Object fields
  roles: set Role, users: set User,
  // Link fields
  ASSIGN: User set->set Role,
}

ASSIGN = ASSIGN:>roles&users<: ASSIGN
```

- Linked objects must exist in the snapshot.

```alloy
AssignRole ( r : Role )
```
Step 2: Transform the Snapshot class to a Snapshot signature containing fields that specify the object configurations within a snapshot.

```alloy
sig Snapshot{
  // Object fields
  roles: set Role, users: set User,
  // Link fields
  ASSIGN: User set->set Role,
}
```

// Linked objects must exist in the snapshot
ASSIGN=ASSIGN:roles&users:<:ASSIGN}
Step 3: Transform each Transition specialization to a predicate in Alloy

pred AssignRole[disj before, after: Snapshot, rPre, rPost: Role, uPre, uPost: User] {
  // Precondition
  rPre in before.roles
  uPre in before.users
  rPre not in uPre.(before.ASSIGN)
  // Postcondition
  rPost in after.roles
  uPost in after.users
  uPost.(after.ASSIGN) = uPre.(before.ASSIGN) + rPost
  // Unchanged object configuration
  ....
}

Snapshot Model to Alloy Model Transformation

- Step 3: Transform each Transition specialization to a predicate in Alloy

Context AssignRole inv:

```
pred AssignRole[disj before, after: Snapshot, rPre, rPost: Role, uPre, uPost: User]{
  // Precondition
  rPre not in uPre.(before.ASSIGN)
  ...
  ...
  // Postcondition
  uPost.(after.ASSIGN) = uPre.(before.ASSIGN) + rPost
  ...
  ...
  // Unchanged object configuration
  after.roles – rPost = before.roles – rPre
  ...
}
```
Snapshot Model to Alloy Model Transformation

- Step 4: Define a trace fact to associate transitions between two consecutive snapshots with operations

```alloy
fact traces{
    all before: Snapshot - SnapshotSequence/last | let after = SnapshotSequence/next[before] | Some r: Role | some u: User | AssignRole[before, after, r, r, u, u]
}
```
Approach Overview
Alloy Instance to UML Object Model Transformation

Alloy Instance

A Sequence of UML Object Models
Research Goal

- Analysis: check whether a system can move from a valid to an invalid state as result of a sequence of access control operation calls
- If analysis uncovers such a sequence, then the designer uses the trace information output by the analysis to help find the source of the errors in the UML policy model
Demonstration Case Study: LRBAC

- Location-aware Role-based Access Control (LRBAC) Model

[Diagram of LRBAC model: User, Session, Role, Location, Object, Operation, relationship arrows showing AssignRole, DeassignRole, ActivateLoc, AddRoleAssignLocation, DeleteRoleAssignLocation, PermAssign, PermRoleLoc, PermObjLoc, PermObj, PermOper]
Demonstration Case Study: LRBAC

- Location-aware Role-based Access Control (LRBAC) Model
Demonstration Case Study: LRBAC

- Location-aware Role-based Access Control (LRBAC) Model
Demonstration Case Study: LRBAC

- Location-aware Role-based Access Control (LRBAC) Model
Demonstration Case Study: LRBAC

- Operation specifications in form of pre/post-conditions for DeleteRoleAssignLocation

  // English specification: remove a location from a set of locations
  // in which a role can be assigned
  Context Role::DeleteRoleAssignLocation(l:Location)
  Precondition: location l has been associated with the role
  Postcondition: location l has been removed from a set of locations

  // OCL specification: remove a location from a set of locations
  // in which a role can be assigned
  Context Role::DeleteRoleAssignLocation(l:Location)
  Pre: self.AssignLoc->includes(l)
  Post: self.AssignLoc=self.AssignLoc@pre->excluding(l)
Demonstration Case Study: Specifying the Property-to-Verify

- Valid and Invalid LRBAC Snapshot Patterns
  - Is there a sequence of operation invocations that takes the system from a valid state consisting of at least one user in a location to an invalid state in which the user is linked to a role that does not include the user’s location?

Demonstration Case Study: Generating the LRBAC Snapshot Model
Demonstration Case Study: Generating an LRBAC Alloy Model

- Role_DeleteRoleAssignLocation predicate generated from the class invariant

```alloy
pred Role_DeleteRoleAssignLocation[disj before,after :Snapshot, rPre, rPost:Role, lPre, lPost:Location] {

// Precondition
Pre in rPre.(before.AssignLoc) …

// Postcondition
rPost.(after.AssignLoc) = rPre.(before.AssignLoc) – lPost…

// Unchanged parts of object configuration
after.roles - rPost = before.roles – rPre …
}
```

- Role_DeleteRoleAssignLocation predicate generated from the class invariant
Demonstration Case Study: Analyzing the Alloy Model

- The verification predicate generated from the property-to-verify

\[
\text{pred valid2invalid}\{}
// Specify that the first snapshot is valid
\text{let first = SnapshotSequence/first} | \ldots
\text{all u:first.users} | u.(first.UserAssign) = \text{none and}
\text{u.(first.UserLoc) \neq none} \ldots
// Query whether there exists a path from a valid
// state to an invalid state
\text{some s: Snapshot - first} | \ldots
\text{I not in r.(s.AssignLoc)} \text{ and}
\text{r in u.(s.UserAssign)} \text{ and I in u.(s.UserLoc)}
\}
Demonstration Case Study: Analyzing the Alloy Model

First Snapshot: Valid

Second Snapshot: Valid
Demonstration Case Study: Analyzing the Alloy Model

Third Snapshot: Valid

Fourth Snapshot: Invalid
Demonstration Case Study: Analyzing the Alloy Model

- Constraint modification: a role can be removed from a set of roles associated with a location only if the role is not assigned to any users

Context Role::DeleteRoleAssignLocation(l:Location)
Pre: self.AssignLoc \(\rightarrow\) includes(l)
and self.UserAssign \(\rightarrow\) isEmpty()
Post: self.AssignLoc = self.AssignLoc@pre \(\rightarrow\) excluding(l)
Conclusion and Future Work

Contribution

- Propose a general framework for, but not limited to, security policy analysis
- Address a usability issue: a usable verification tool needs to be integrated with the UML-based development processes and notations used by software developers

Future work

- Investigating how safety and liveness access control properties can be analyzed using model-checking techniques at the back-end in a usable manner

*Acknowledgment: the work was supported by the National Science Foundation grant CCF-1018711.
Background: Snapshot Model

- Example of a snapshot model generated from a class model
Background: Snapshot Model

Operation specification for AssignRole(r:Role) in the original design class model:

// English specification: assign a role to a user
Context User::AssignRole(r:Role)
Precondition: role r is not assigned to the user
Postcondition: role r is assigned to the user

// OCL specification: assign a role to a user
Context User::AssignRole(r:Role)
Pre: self.ASSIGN->excludes(r)
Post: self.ASSIGN = self.ASSIGN@pre->including(r)
Example of the specialized class invariant in a snapshot model generated from the operation specification in a design class model:

Context AssignRole inv:

// Generated from precondition of User::AssignRole(r:Role)
// role r is not assigned to the user
uPre.ASSIGN->excludes(rPre) and ...

// Generated from postcondition of User::AssignRole(r:Role)
// role r is assigned to the user
uPost.ASSIGN = uPre.ASSIGN->including(rPost) and ...

// Unchanged parts of object configuration
after.roles->excluding(rPost)=before.roles->excluding(rPre)
...

Background: Snapshot Model

- Example of the specialized class invariant in a snapshot model generated from the operation specification in a design class model:

  Context AssignRole inv:
  // Generated from precondition of User::AssignRole(r:Role)
  // role r is not assigned to the user
  uPre.ASSIGN->excludes(rPre) and …

  // Generated from postcondition of User::AssignRole(r:Role)
  // role r is assigned to the user
  uPost.ASSIGN = uPre.ASSIGN->including(rPost) and …

  // Unchanged parts of object configuration
  after.roles->excluding(rPost)=before.roles->excluding(rPre)
  …
Example of the specialized class invariant in a snapshot model generated from the operation specification in a design class model:

Context AssignRole inv:
// Generated from precondition of User::AssignRole(r:Role)
// role r is not assigned to the user
uPre.ASSIGN->excludes(rPre) and …

// Generated from postcondition of User::AssignRole(r:Role)
// role r is assigned to the user
uPost.ASSIGN = uPre.ASSIGN->including(rPost) and …

// Unchanged parts of object configuration
after.roles->excluding(rPost)=before.roles->excluding(rPre)
…
Snapshot Model to Alloy Model Transformation

- Step 2: Transform the Snapshot class to a Snapshot signature containing fields that specify the object configurations within a snapshot.

```alloy
sig Snapshot{
  // Object fields
  roles: set Role, users: set User,
  // Link fields
  ASSIGN: User set->set Role,
}
```

// Linked objects must exist in the snapshot
ASSIGN=ASSIGN:>roles&users<: ASSIGN

```alloy
AssignRole ( r : Role )
```

```alloy
Snapshot
```

```alloy
Role
```

```alloy
User
```

```alloy
Transition
```

```alloy
AssignRole ( r : Role )
```
Demonstration Case Study: Generating the LRBAC Snapshot Model

- Invariant for Role_DeleteRoleAssignLocation class generated from the specification of DeleteRoleAssignLocation operation

  Context Role_DeleteRoleAssignLocation inv:
  // Generated from precondition of DeleteRoleAssignLocation
  rPre.AssignLoc->includes(lPre)

  ...

  // Generated from postcondition of DeleteRoleAssignLocation
  rPost.AssignLoc=rPre.AssingLoc->excluding(lPost) and

  ...

  // Unchanged parts of object configuration
  after.roles->excluding(rPost)
  =before.roles->excluding(rPre) and

  ...

© Colorado State University
Computer Science Department
Alloy Instance to UML Object Model Transformation

Step 1

Alloy Instance in XML

Step 2

Object Models in XMI

Step 3

Class Model in XMI

Snapshots:
- Snapshot0: users, User0, NULL
- roles, Role0, User1
- Snapshot1: users, User0, Role1
- roles, Role0, NULL
- sessions, Session0, NULL

Field Labels:
- users
- roles
- sessions

Atom Labels:
- User0, Role0, NULL
- User1, Role1

XMI Representation