A Case Study Details: Test Generation for Wind Turbine Access Control Policies

A.1 Wind Turbine Access Control Requirements

For our case studies, we considered security requirements for the WT domain found in literature. Below we summarize the requirements in [55,7], reinpreted over the exact metamodel (see Figure 2) used in the case study, and slightly edited for consistency, to serve as the basis of implementing the policy.

- Users of the modeling platform consist of the unrestricted user `Owner` and the group of `RestrictedUsers`. The latter group is composed of users `IOManager`, `WTManager`, `SystemManager` and the sub-group `Specialists`. For each kind of specialization (see Figure 2), there is a corresponding user in the latter group, namely `FanEngineer`, `HeaterEngineer`, and `PumpEngineer`.
- The user `Owner` is a super-user overseeing the collaborative effort. They can add/remove/edit all kinds of content, including authorization information (i.e. `Ownership`, `protected`).
- The user `IOManager` works only with external inputs and outputs of the system (of type `SystemInput` resp. `SystemOutput`); they only have an obfuscated view of the `WT` system and have no access to any other element in the model.
- The user `WTManager` can modify the `WT` system and freely edit `ControlUnit`s; as well as reference `SystemInput` and `SystemOutput` elements in a read-only way; but cannot see other kinds of elements.
- The user `SystemManager` can modify the `WT` system and freely edit `ControlUnit`s; as well as reference other kinds of control elements (but not authorization information) in a read-only way.
- Specialists have full access to `Subsystem`s they own and their contents, and can read or write signals directly provided as outputs of control units transitively contained in these subsystems.
- As an exception to the above, specialists cannot modify `SystemInput` and `SystemOutput` elements.
- Specialists can additionally see (in a read-only way) `Subsystem`s transitively containing their owned `Subsystem`s, as well as signals directly provided (a) either as outputs or internal variables of these containing subsystems, or (b) non-internal output variables of sibling subsystems under the same container.
- As an exception to all of the above, no specialist is allowed to see a subsystem marked as `protected`.
- Specialists have no read access to any other model element.

We have implemented an access control policy that expresses these access requirements in the *MONDO Access Control Policy* language. This baseline policy is included in Section A.2 (along with supporting model queries in Section A.3), while mutants generated from it are discussed in Section A.4.

A.2 Baseline access control policy for the Wind Turbine domain

```
import "policyQueries.vql"

user IOManager
user SystemManager
user WTManager
user Owner

user PumpEngineer
user HeaterEngineer
user FanEngineer
group Specialists {
  FanEngineer,HeaterEngineer,PumpEngineer
}

group RestrictedUsers {
  Specialists, IOManager, SystemManager, WTManager
}

policy WindTurbine deny by default {

  //IOManager is allowed to work with IO
  rule permitIO allow RW to IOManager {
    from query "externalIO"
    select obj(io)
  } with 200 priority
  //IOManager is not allowed to work with Ctrls, subsystems
  rule hideModules deny RW to IOManager {
    from query "objectModules"
    select obj(module)
  } with 190 priority
  rule hideControls deny RW to IOManager {
    from query "objectControls"
    select obj(ctrl)
  } with 190 priority
  //Attributes of WT System are obfuscated from IOManager
  rule obfSystem obfuscate to IOManager {
    from query "wtRoot"
    select obj(system)
  } with 180 priority

  //WTManager, WTSystemManager, Specialists are not allowed to modify IOs
  rule readOnlyIO deny W to WTManager, SystemManager, Specialists {
    from query "externalIO"
    select obj(io)
  } with 140 priority
  //IOManager, WTManager are not allowed to see and modify Misc elements
  rule hideMisc deny RW to IOManager, WTManager {
    from query "objectMisc"
    select obj(misc)
  } with 130 priority
  // SystemManager are not allowed to modify Misc elements
  rule readOnlyMisc deny W to SystemManager {
    from query "objectMisc"
    select obj(misc)
  } with 120 priority


  // Specialists cannot see protected modules
  rule hideDescriptionOfProtected deny RW to Specialists {
    from query "moduleProtected"
    select obj(module)
  } with 100 priority
  rule hideConsumptionByProtected deny RW to Specialists {
    from query "consumptionProtected"
```

```
  select ref(ctrl->signal:inputs)
} with 100 priority

//Specialists have full access to their own modules
rule ownModulesFAN allow RW to FanEngineer {
  from query "moduleOwnedBySpecialist"
  select obj(module)
  where specialist is bound to ::FAN
} with 90 priority
rule ownModulesHEATER allow RW to HeaterEngineer {
  from query "moduleOwnedBySpecialist"
  select obj(module)
  where specialist is bound to ::HEATER
} with 90 priority
rule ownModulesPUMP allow RW to PumpEngineer {
  from query "moduleOwnedBySpecialist"
  select obj(module)
  where specialist is bound to ::PUMP
} with 90 priority

//Specialists also have full access to signals provided by their own modules
rule ownSignalsFAN allow RW to FanEngineer {
  from query "signalOwnedBySpecialist"
  select obj(signal)
  where specialist is bound to ::FAN
} with 90 priority
rule ownSignalsHEATER allow RW to HeaterEngineer {
  from query "signalOwnedBySpecialist"
  select obj(signal)
  where specialist is bound to ::HEATER
} with 90 priority
rule ownSignalsPUMP allow RW to PumpEngineer {
  from query "signalOwnedBySpecialist"
  select obj(signal)
  where specialist is bound to ::PUMP
} with 90 priority

//Specialists have read access to containing modules
rule containingModulesFAN allow R to FanEngineer {
  from query "moduleVisibleToSpecialistSubmodule"
  select obj(container)
  where specialist is bound to ::FAN
} with 90 priority
rule containingModulesHEATER allow R to HeaterEngineer {
  from query "moduleVisibleToSpecialistSubmodule"
  select obj(container)
  where specialist is bound to ::HEATER
} with 90 priority
rule containingModulesPUMP allow R to PumpEngineer {
  from query "moduleVisibleToSpecialistSubmodule"
  select obj(container)
  where specialist is bound to ::PUMP
} with 90 priority

//Specialists have read access to non-internal signals of sibling modules
rule siblingSignalsFAN allow R to FanEngineer {
  from query "signalVisibleToSpecialistSubmodule"
  select obj(signal)
  where specialist is bound to ::FAN
} with 90 priority
rule siblingSignalsHEATER allow R to HeaterEngineer {
  from query "signalVisibleToSpecialistSubmodule"
  select obj(signal)
  where specialist is bound to ::HEATER
} with 90 priority
rule siblingSignalsPUMP allow R to PumpEngineer {
```

```
    from query "signalVisibleToSpecialistSubmodule"
    select obj(signal)
    where specialist is bound to ::PUMP
  } with 90 priority

  // specialists have no knowledge of the rest
  rule hideSystem deny RW to Specialists {
    from query "wtRoot"
    select obj(system)
  } with 80 priority

  // Rules governing the authorization itself
  rule hideDefault deny RW to RestrictedUsers {
    from query "modelRoot"
    select obj(root)
  } with 20 priority
  rule fullAccess allow RW to Owner {
    from query "modelRoot"
    select obj(root)
  } with 10 priority

} with restrictive resolution
```

A.3 Model queries (graph patterns) used by the policy, in VIATRA QUERY syntax

```
package hu.bme.mit.sttt18.wt.policy

import epackage "http://WTSpec4M/5.0"
import epackage "http://www.eclipse.org/emf/2002/Ecore"

// Queries all system-level input and output objects in the model
pattern externalIO(io : EObject) {
  SystemOutput(io);
} or {
  SystemInput(io);
}
// Queries all controller objects in the model
pattern objectControls(ctrl : ControlUnit) {
  ControlUnit(ctrl);
}
// Queries the root system object
pattern wtRoot(system : WT) {
  WT(system);
}
// Queries the root model object
pattern modelRoot(root : AuthorizedSystem) {
  AuthorizedSystem(root);
}

// All subsystems (incl. main subsystems)
pattern objectModules(module : Subsystem) {
  Subsystem(module);
}

// Queries all miscellaneous objects in the model
pattern objectMisc(misc : EObject) {
  WTCParam(misc);
} or {
  WTCTimer(misc);
} or {
  WTCFault(misc);
} or {
  SystemVariable(misc);
}

// Subsystems marked as protected IP
pattern moduleProtected(module : Subsystem) {
  AuthorizedSystem.protected(_, module);
}
// Inputs read by control units in protected IP subsystems
pattern consumptionProtected(ctrl : ControlUnit, signal : WTCInput) {
  find moduleProtected(module);
  find subsystems+(module, submodule);
  Subsystem.controlUnits(submodule, ctrl);
  ControlUnit.inputs(ctrl, signal);
} or {
  find moduleProtected(module);
  module == submodule;
  Subsystem.controlUnits(submodule, ctrl);
  ControlUnit.inputs(ctrl, signal);
}
// Composite subsystems and sub-subsystems
pattern subsystems(module, submodule) {
  Subsystem.subsystems(module, submodule);
}

// Subsystem assigned to given specialist
pattern moduleOwnedBySpecialist(module : Subsystem, specialist : SpecialistKind) {
  Ownership.owned(o, module);
```

```
  Ownership.owner(o, specialist);
}
// Output or internal signal of a subsystem assigned to a specialist
pattern signalOwnedBySpecialist(module : Subsystem, signal : WTCOutput, specialist : SpecialistKind) {
  find providedByModule(module, signal);
  find moduleOwnedBySpecialist(module, specialist);
}
// Transitive container of a subsystem assigned to a specialist
pattern moduleVisibleToSpecialistSubmodule(container : Subsystem, specialist : SpecialistKind) {
  find subsystems+(container, submodule);
  find moduleOwnedBySpecialist(submodule, specialist);
}
// Outputs or internal signals of transitively containing submodules, as well as non-internal outputs of sibl
pattern signalVisibleToSpecialistSubmodule(container : Subsystem, signal : WTCOutput, specialist : Specialist
  find moduleVisibleToSpecialistSubmodule(container, specialist);
  find directlyProvidedByModule(container, signal);
} or {
  find moduleVisibleToSpecialistSubmodule(container, specialist);
  Subsystem.subsystems(container, sibling);
  find providedByModuleNonInternal(sibling, signal);
}

// Non-internal outputs of subsystems
pattern providedByModuleNonInternal(module : Subsystem, signal : WTCOutput) {
  find providedByModule(module, signal);
  neg find consumedByModule(module, signal);
}

// Output or internal signal of a subsystem (provided by transitively contained control unit)
pattern providedByModule(module : Subsystem, signal : WTCOutput) {
  find subsystems+(module, submodule);
  find directlyProvidedByModule(submodule, signal);
} or {
  module == submodule;
  find directlyProvidedByModule(submodule, signal);
}
// Output or internal signal of a subsystem (provided by directly contained control unit)
pattern directlyProvidedByModule(module : Subsystem, signal : WTCOutput) {
  Subsystem.controlUnits.outputs(module, signal);
}
// Input or internal signal of a subsystem (consumed by transitively contained control unit)
pattern consumedByModule(module : Subsystem, signal : WTCInput) {
  find subsystems+(module, submodule);
  find directlyConsumedByModule(submodule, signal);
} or {
  module == submodule;
  find directlyConsumedByModule(submodule, signal);
}
// Input or internal signal of a subsystem (consumed by directly contained control unit)
pattern directlyConsumedByModule(module : Subsystem, signal : WTCInput) {
  Subsystem.controlUnits.inputs(module, signal);
}
```

A.4 Overview of policy mutants

As discussed in Section 4.1, we applied mutation operators to introduce changes to any part of the access control policy, *except* the underlying model queries and the *binding* of these queries to individual rules. The mutation operators were similar in spirit to those found in [9,35], but adapted to the *MONDO Access Control* language instead of XACML.

In the following, we list each of the applied mutation operators. In parentheses, we indicate the number of times the individual operator was applied. Altogether 174 mutant policies were derived.

Remove Rule (24) Take an access rule and remove it from the policy.

Add Subject to Rule (111) Take an access rule and add a subject (user or group) to the list of subject affected by rule. Applicable only if the subject is not currently listed for the rule, nor is it contained in any group listed for the rule.

Remove Subject from Rule (4) Take an access rule and remove an entry from the list of subjects affected by rule. Applicable only if the list doeas not become empty (as there is a separate operator for removing a rule entirely).

Toggle Rule Access Level (24) Take an access rule and change the granted access level (allow/obfuscate/deny).

Toggle Rule Operation (8) Take an access rule that controls read permissions and switch it to control write permissions (for the same subject and access level), or vice versa. Rules that jointly control read and write permissions are unaffected (as there is a separate operator for removing a rule entirely).

Exchange Priority (3) Take two different priority classes for access rules, and two different access levels (allow/obfuscate/deny), and switch all rules from the first class and first level to the second class and second level, and vice versa. Applicable only if there are at least one rule affected in both directions.

A.5 Well-formedness constraints

```
package windturbine.queries
import "http://WTSpec4M/5.0"

@Constraint(key={M}, message="MainSubsystem can not be contained by another MainSubsystem", severity="error")
pattern containedMainSubsystem(M:MainSubsystem) {
  Subsystem.subsystems(_,M);
}

@Constraint(key={I}, message="Inputs must be consumed by at least one control unit", severity="error")
pattern unusedInput(I:WTCInput){
  neg find usedInput(_,I);
}

pattern usedInput(C:ControlUnit,I:WTCInput) {
  ControlUnit.inputs(C,I);
}

@Constraint(key={O}, message="Outputs must be provided by at least one control unit", severity="error")
pattern unusedOutput(O:WTCOutput){
  neg find usedOutput(_,O);
}

@Constraint(key={O}, message="Outputs must be provided by at most one control unit", severity="error")
pattern collidingOutput(O:WTCOutput){
  find usedOutput(c1, O);
  find usedOutput(c2, O);
  c1 != c2;
}


pattern usedOutput(C:ControlUnit,O:WTCOutput) {
  ControlUnit.outputs(C,O);
}

@Constraint(key={T}, message="Timers must be used by at least one control unit", severity="error")
pattern unusedTimer(T:WTCTimer) {
  neg find usedTimer(_,T);
}

pattern usedTimer(C:ControlUnit,T:WTCTimer) {
  ControlUnit.timers(C,T);
}

@Constraint(key={F}, message="Faults must be used by at least one control unit", severity="error")
pattern unusedFault(F:WTCFault) {
  neg find usedFault(_,F);
}

pattern usedFault(C:ControlUnit,F:WTCFault){
  ControlUnit.fault(C,F);
}

@Constraint(key={P}, message="Params must be used by at least one control unit", severity="error")
pattern unusedParam(P:WTCParam) {
  neg find usedParam(_,P);
}

pattern usedParam(C:ControlUnit,P:WTCParam) {
  ControlUnit.params(C,P);
}

@Constraint(key={C}, message="ControlUnits must have at least one input or output", severity="error")
pattern uselessControlUnit(C:ControlUnit) {
  neg find usedInput(C,_);
```

```
  neg find usedOutput(C,_);
}

@Constraint(key={S}, message="Subsystems must have at least one control unit or subsystem", severity="error")
pattern uselessSubsystem(S:Subsystem) {
  neg find subSubsystem(S,_);
  neg find subsystemWithControlUnit(S,_);
}

pattern subSubsystem(S1:Subsystem,S2:Subsystem) {
  Subsystem.subsystems(S1,S2);
}

pattern subsystemWithControlUnit(S:Subsystem,C:ControlUnit) {
  Subsystem.controlUnits(S,C);
}

@Constraint(key={s}, message="Subsystems must be owned by at most one specialist", severity="error")
pattern collidingOwnership(s: Subsystem){
  Ownership.owned(o1, s);
  Ownership.owned(o2, s);
  o1 != o2;
}
```