Specification Of Access Control Features In Health Records

Sai Venkatesh (IMT2017012)
George Abraham (IMT2017019)
Kaushal Mittal (IMT2017024)

Outline

- What is Athena ?
- Basic introduction to Athena.
- Athena as a Theorem Prover.
- A Basic Approach to Access Control Policy.
- What is SMT solver ?
- A more expressive approach (“ Sophisticated Access Control via SMT and Logical Frameworks ”)
- Conclusion.

What is Athena ?

Athena as a programming language

- Higher Order
- Dynamically Typed
- Allows Side effects

Athena as a logic tool (Theorem Prover)

- A programmable theorem-proving system for logical statements.

Introduction to Athena

- Domain
    - It is a set of objects of a certain *sort* (type).

```
>domain Person;;

New domain Person introduced.
```

- Function Symbols

```
>(declare mother (-> (Person) Person));;

New symbol mother declared.
```

Introduction to Athena

- Function Symbols

```
>(declare mother (-> (Person) Person));;

New symbol mother declared.
```

- **Function Symbols ≠ Functions**

```
>(mother Joe = Alice);;

Term: (= (mother Joe)
         Alice)

>(mother Joe);;

Term: (mother Joe)
```

Introduction to Athena

- Terms
  - Elements of some **domain**
  - **Variable** belonging to some domain.
  - Terms can be formed by applying **function symbols** to other **terms**

```
>Joe;;

Term: Joe

>(mother ?x);;

Term: (mother ?x:Person)

>(mother Joe);;

Term: (mother Joe)
```

Introduction to Athena

- Sentences
  - Terms of sort Boolean
  - Boolean combinations of other sentences: **and, or, if, iff, forall, exists**

```
>(forall ?x . ?x =/= mother ?x);;

Sentence: (forall ?x:Person
            (not (= ?x:Person
                    (mother ?x:Person))))
```

Athena as a Theorem Prover

- Assumption-base
  - Global set of sentences, that regard as true (premices/axioms).
- Assert
  - Is a procedure which inserts sentences into assumption-base.

```
>(assert (mother Joe = Alice));;

The sentence
(= (mother Joe)
   Alice)
has been added to the assumption base.

>(show-assumption-base);;

There is one sentence in the assumption base:

(= (mother Joe)
   Alice)
```

Athena as a Theorem Prover

- Manual method
  - **Primitive methods:** Inference rules (eg addition, simplification, modus ponen)
  - Combination of multiple primitive methods to form bigger proof
  - Eg **(and A B)** => **(and B A)**

```
>(assert (and A B));;

The sentence
(and A B)
has been added to the assumption base.

>(dseq (!left-and (and A B))
       (!right-and (and A B))
       (!both B A));;

Theorem: (and B A)
```

Athena as a Theorem Prover

- Automated method

```
>(assert (and A B));;

The sentence
(and A B)
has been added to the assumption base.

>(!prove (and B A) (get-assumption-base));;

Theorem: (and B A)
```

A Basic approach for Access Control Policy

Characteristics of Hospital and Mapping them in Athena:

- What are the **domains (Doctors ,Patients)**?
- **Resources** are the medical records.
- Every doctor belongs to a department.
- Every patient has a consulting doctor (implies that Patient's record is a part of corresponding doctor's department).
- **Relation between doctors(Some doctors report to other doctors)**

```
#declaring domains doctors and patients
domain Patient
domain Doctor

declare patient1,patient2 : Patient
declare doc1,doc2,doc3 : Doctor
```

Question we are interested in solving is whether a particular medical professional can access a specific patient's health record?

Code details

Keeping the previous policies in mind, we use ATP to **find if there are violations in the governing Policies** regarding the privacy of patient.

Instantiating primitives :

```
declare doctor_of : [Patient] -> Doctor
declare can_access : [Patient Doctor]-> Boolean
declare reports : [Doctor Doctor] -> Boolean
```

- **doctor_of : [Patient] ->Doctor** (patient as input and gives out doctor of that patient)
- **can_access: [Patient Doctor] -> Boolean** (Boolean value if doctor can access a patient’s record)
- **reports: [Doctor Doctor] -> Boolean** (Boolean whether argument 1 reports to argument 2)

Governing Policies

Policy 1:

Talks about the ‘**reports to**’ relation.If doctor : ‘w1’ reports to doctor :’w2’ , denoted by $w1 \rightsquigarrow w2$.

Rules governing Policy1:

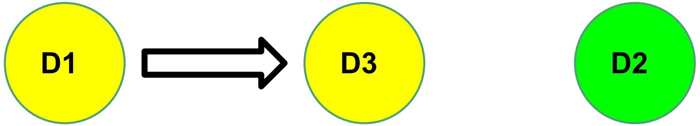
- $\rightsquigarrow$ relation is transitive. $(w1 \rightsquigarrow w2) \cap (w2 \rightsquigarrow w3) \Rightarrow (w1 \rightsquigarrow w3)$
- $\rightsquigarrow$ relation is antisymmetric. $(w1 \rightsquigarrow w2) \Rightarrow \neg (w2 \rightsquigarrow w1)$
- $(w1 \rightsquigarrow w2) \nRightarrow$ department (w1) = (department w2)
- $(w1 \rightsquigarrow w2) \Rightarrow$ can_access (w1) $\subseteq$ can_access (w2)

Policy 2:

This policy relates to the privacy of patient.Records cannot be accessed by doctors from different departments unless allowed by the patient owning the record.

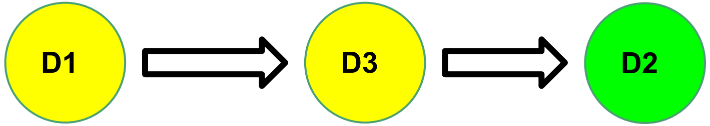
Inconsistencies Arising from the Policies

If we observe closely , there is a possible inconsistency arising when we put both the Policies together.What is it?



(Case 1: Consistent case)

```
Unable to derive the conclusion
(exists ?y:Doctor
  (exists ?z:Doctor
    (exists ?h:Patient
      (or (and (in_Dep1 ?y:Doctor)
               (in_Dep2 ?z:Doctor)
               (reports ?y:Doctor ?z:Doctor)
               (= (doctor_of ?h:Patient)
                  ?y:Doctor)
               (can_access ?h:Patient ?z:Doctor))
          (and (in_Dep2 ?y:Doctor)
               (in_Dep1 ?z:Doctor)
               (reports ?y:Doctor ?z:Doctor)
               (= (doctor_of ?h:Patient)
                  ?y:Doctor)
               (can_access ?h:Patient ?z:Doctor))))))
```



(Case 2: Inconsistent case, because of cross department reporting)

```
Theorem: (exists ?y:Doctor
           (exists ?z:Doctor
             (exists ?h:Patient
               (or (and (in_Dep1 ?y:Doctor)
                        (in_Dep2 ?z:Doctor)
                        (reports ?y:Doctor ?z:Doctor)
                        (= (doctor_of ?h:Patient)
                           ?y:Doctor)
                        (can_access ?h:Patient ?z:Doctor))
                   (and (in_Dep2 ?y:Doctor)
                        (in_Dep1 ?z:Doctor)
                        (reports ?y:Doctor ?z:Doctor)
                        (= (doctor_of ?h:Patient)
                           ?y:Doctor)
                        (can_access ?h:Patient ?z:Doctor))))))
```

Conclusions from the basic approach

Advantages:

- Can be used to find if there is instantiation of violation in any policy.
- Can perform well on query evaluation made by the users

Limitations:

- Checking whether there is **redundancy** of a policy P (i.e if addition of P to the set of policies S does not add any observable change.) is quite difficult
- Unable to find if a particular **policy is empty** (does not allow any requests) (or) whether the policy allows all requests (**completeness**)
- It is unable to point to which property of a policy leads to the **inconsistency** and reduces our ability for corrective measures.

A more expressive approach

SMT (Satisfiability Modulo Theories)

- What is SMT?
- Input and Outputs of a SMT
- First-Order Logic
- Eg - Logical Symbols - $\neg$, $\vee$, $\wedge$, $\forall$, $\exists$ and Parameters - =, +, > , < , constants.
- Interfacing SMT on Athena

Why use SMT over Automated Theorem Proving and other advantages.

Access Control via SMT and Logical Frameworks

[KONSTANTINE ARKOUDAS, RITU CHADHA, and JASON CHIANG, ACM-2014]

- Formulating, Analyzing and applying access-control policies.
- They reduce both request evaluation and policy analysis to SMT solving

Goals that SMT helps to achieve when analysing policies

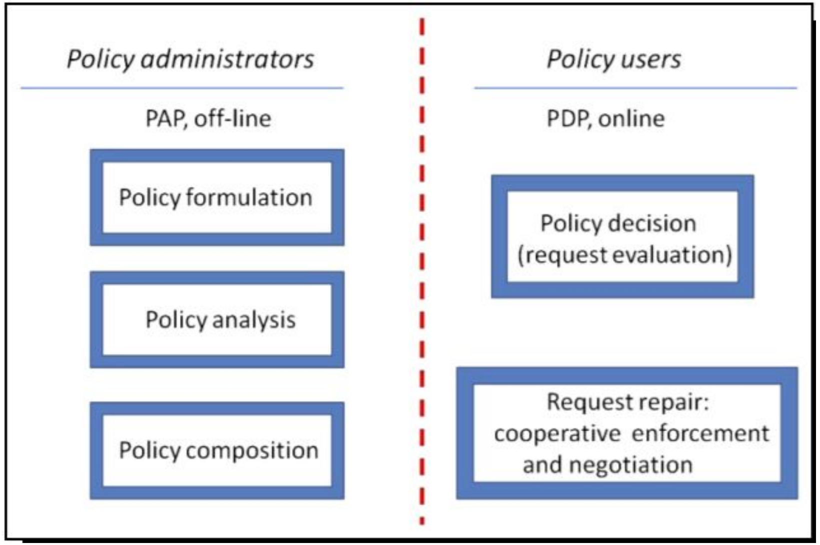
- Consistency
- Coverage
- Redundancy
- Change Impact

Roles of Policy administrators (functionality)

- Expressively formulate policies using first-order logic.
- Analysing formulated policies

Roles of Policy User

- Request Evaluation
- Request Repair or Co-operative enforcement



Policy Formulation

In this framework, they have defined a request such that they consist of

- Subjects (or “principals”): Who initiated the request
- Objects (or “resources”): What resource they want to access
- Actions (or “rights”): What they want to do with the resource
- Request : Raised by a Subject to perform certain Action over certain Object.

Function symbol to handle requests

- subject: **Request → Subject**
- object : **Request →Object**
- action: **Request → Action**
- allow, deny : **Request → Boolean**

Policy Analysis

define verify := λ P, property . let property' = constraintsOf(P) ∧ ¬property(P)
m = findModel(property'),
if m == none => (property hold), else m is counterexample

- Consistency : A policy is consistent iff, for any given request r, r is either denied or permitted (exclusively)

$$\lambda P . verify(P, \lambda P . \lambda r . \neg(permits(P, r) \wedge denies(P, r)))$$

- Coverage : Check if every request is either permitted or denied

  λP . verify(P, λP . λ r . ¬covers(P, r))
- Redundancy : Pi is redundant => f([P1 ··· Pn]) ~= f([P1 ··· Pi−1 Pi+1 ··· Pn])
- Change Impact : To avoid unintended consequences.

Request Evaluation

- Policy **P**
- Request **Q(t)**
- Conjunction of all of **P's** environmental constraints **C(t)**
- Permissive predicates of P **A(t)**
- Prohibitive predicates of the basis of P **D(t)**

Possible Scenarios are 'permit, 'deny or 'na

- If **Q(t)** $\wedge$ **C(t)** $\wedge\neg$**A(t)** is unsatisfiable, then return 'permit
- Else if, **Q(t)** $\wedge$ **C(t)** $\wedge\neg$**D(t)** is unsatisfiable, return 'deny
- Else, return 'na

Or other way*

Structure of Policies

- a tag identifying the object as an atomic policy/composite policy
- a name (optional), represented by a string
- a rule base (A rule base consists of a list of access rules.)
- a rule integrator (rule integrator is a procedure that takes an arbitrary rule base and produces a policy basis using an integrating rule)
- a list of environmental constraints
- a policy basis. (defined as a list of two predicates [A D] .The first predicate A, characterizes all and only those requests that are allowed by the corresponding policy; the second predicate, D, characterizes the requests that are denied.)