Access Control Policy Mining:
RBAC to ABAC

L10-2
CS6393
Spring 2020

**** Role Mining and ABAC policy mining references can be found in “Related works” section of paper 2
Role-Based Access Control (RBAC) assigns user to “role”
Paper 2 Workflow

(a) Given RBAC system only

Check ABAC RuleSet Existence (partition-based approach)

Yes

(b) Given RBAC system with supporting data

Infeasibility correction (partition-based approach) (use role-based attributes)

No

Rule Generation

@Shuvra Chakraborty
• RBAC (Role-Based Access Control) is widely used but has notable limitations (e.g., role explosion)
• Using ABAC (Attribute-Based Access Control), access control policies can be written in more flexible and higher level way
• Automated migration of an existing RBAC system to ABAC system (defined as ABAC policy mining problem) cuts the cost and human efforts needed
• Stoller et. al. use explicit unique IDs in attribute set to resolve ABAC policy mining problem which is somehow conflicting with basic principle of ABAC

- We introduce ABAC RuleSet Existence problem: questions the feasibility of ABAC policy mining problem in RBAC context
  - If not feasible without ID, infeasibility correction technique is applied
  - Eliminates use of explicit ID in ABAC policy mining
1. Access control

2. An Access control system must have a checkAccess function which evaluates an access request (user, object, operation) to true/false

3. Two access control systems are equivalent iff i) set of users (U), objects (O), and operations (OP) are identical ii) for any access request, checkAccess_{system1} and checkAccess_{system2} evaluates the same

4. The following example includes 3 types of Access Control System
   a. Enumerated Authorization System (EAS)
   b. RBAC System
   c. ABAC System
EAS is a tuple <U, O, OP, AUTH, checkAccess_EAS>

- U, O, and OP are finite sets of users, objects and operations, respectively
- AUTH ⊆ U × O × OP

**Example 1:**
- U = {John, Lina, Ray, Tom}, OP = {read, write}, O = {Obj1, Obj2}

<table>
<thead>
<tr>
<th>AUTH</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(John, Obj1, write)</td>
<td>e.g., John is allowed to do read operation on Obj1 but not allowed to do read operation on Obj2</td>
</tr>
<tr>
<td>(John, Obj2, write)</td>
<td></td>
</tr>
<tr>
<td>(John, Obj1, read)</td>
<td></td>
</tr>
<tr>
<td>(Lina, Obj2, write)</td>
<td></td>
</tr>
<tr>
<td>(Tom, Obj1, read)</td>
<td></td>
</tr>
<tr>
<td>(Ray, Obj1, read)</td>
<td></td>
</tr>
</tbody>
</table>
RBAC system
- is a tuple \(<U, O, OP, \text{Roles}, \text{RPA}, \text{RUA}, \text{RH}, \text{checkAccess}_{\text{RBAC}}>\)

- Roles is a finite set of roles
- RH is the role hierarchy relation \([\text{RH'}: \text{reflexive transitive closure of RH}]\)
- RPA : Role Permission Assignment
- RUA: Role User Assignment
- Permission is an object-operation pair
- \(\text{authPerm}(r) = \{p \in \text{RPA}(r')|(r, r') \in \text{RH'}\}, \text{where } r,r' \in \text{Roles}\)
- \(\text{authUser} (r) = \{u \in \text{RUA}(r')|(r', r) \in \text{RH'}\} \text{ where } r,r' \in \text{Roles}\)

- \(\text{checkAccess}_{\text{RBAC}}(u:U, o:O, op:OP) \equiv \exists r \in \text{Roles}.(u \in \text{authUser}(r) \land p \in \text{authPerm}(r) \land (o, op) = (\text{obj}(p), \text{ops}(p)))\)
Example 2:

- U = {John, Lina, Ray, Tom}, OP = {read, write}, O = {Obj1, Obj2} [same as Example 1]
- Roles = {R1, R2, R3}
- RPA(R1) = {(Obj1, write)}, RPA(R2) = {(Obj2, write)}, RPA(R3) = {(Obj1, read)}
- RUA(R1) = {John}, RUA(R2) = {Lina}, RPA(R3) = {Ray, Tom}
- RH = {(R1, R2), (R1, R3)} [R1 is a senior role than R2, R3]

EAS and RBAC system defined in example 1 and 2 are equivalent
ABAC system is a tuple \(<U, O, OP, UA, OA, UAValue, OAValue, RangeSet, RuleSet, checkAccess_{ABAC}>\)

**Example 3**

- U, O, OP are same as Example 1
- UA =\{Position, Dept.\}, OA = \{Type\}

<table>
<thead>
<tr>
<th>User (U)</th>
<th>Position</th>
<th>Dept.</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Officer</td>
<td>CS</td>
</tr>
<tr>
<td>Lina</td>
<td>Student</td>
<td>CS</td>
</tr>
<tr>
<td>Ray</td>
<td>Officer</td>
<td>CS</td>
</tr>
<tr>
<td>Tom</td>
<td>Officer</td>
<td>CS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RangeSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
<tr>
<td>{Officer, Student, Faculty}</td>
</tr>
<tr>
<td>Dept.</td>
</tr>
<tr>
<td>{CS, EE}</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>{File, Printer, Scanner}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OAValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Obj1</td>
</tr>
<tr>
<td>Obj2</td>
</tr>
</tbody>
</table>

- RuleSet contains one separate rule for each operation, \{Rule_{read}, Rule_{write}\}
- **ABAC system is incomplete in Example 3 (No rules given!)**

@Shuvra Chakraborty
ABAC rule structure
For any operation \( op \in OP \), \( Rule_{op} \) grammar

- \( Rule_{op} ::= Rule_{op} \lor Rule_{op} \mid (Atomicexp) \)
- \( Atomicexp ::= Atomicuexp \land Atomicoexp \mid Atomicuexp \mid Atomicoexp \)
- \( Atomicuexp ::= Atomicuexp \land Atomicuexp \mid uexp \)
- \( Atomicoexp ::= Atomicoexp \land Atomicoexp \mid oexp \)
- \( uexp \in \{ua(u) = value \mid ua \in UA \land value \in Range(ua)\} \)
- \( oexp \in \{oa(o) = value \mid oa \in OA \land value \in Range(oa)\} \)

- \( checkAccess_{ABAC} (a:U, b:O, op:OP) \equiv Rule_{op}(a:U, b:O) \)

*** Illustrated ABAC rule examples can be found in later slides
*** Example 1,2 and 3 will be used to demonstrate the workflow of paper 2
ABAC RuleSet Existence

Does an equivalent ABAC system exist for the given RBAC system and supporting data?

Find the RuleSet -> *With ID, always possible, *No IDs → Not possible

e.g., cannot separate John from Ray and Tom in Example 3

@Shuvra Chakraborty
Step 1. Generate role-based attribute set

- For a user $u$, role-based user attribute denotes the set of roles possessed by $u$
- For an object-operation pair $(obj, op)$, role-based object attribute denotes the set of roles where each role contains permission $(obj, op)$

### Role-based user attribute (Example 2)

<table>
<thead>
<tr>
<th>User(U)</th>
<th>uroleAtt</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>${R1, R2, R3}$</td>
</tr>
<tr>
<td>Lina</td>
<td>${R2}$</td>
</tr>
<tr>
<td>Ray</td>
<td>${R3}$</td>
</tr>
<tr>
<td>Tom</td>
<td>${R3}$</td>
</tr>
</tbody>
</table>

### Role-based object attribute (Example 2)

<table>
<thead>
<tr>
<th>Object(O)</th>
<th>oroleAtt$_{write}$</th>
<th>oroleAtt$_{read}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obj1</td>
<td>${R1}$</td>
<td>${R1, R3}$</td>
</tr>
<tr>
<td>Obj2</td>
<td>${R1, R2}$</td>
<td>${}$</td>
</tr>
</tbody>
</table>

Next step: partition set is generated on set $U\times O$ based on similarity in attribute value assignment
Step 2

Partition set w.r.t. write

Ray, Obj1
Tom, Obj1
Ray, Obj2
Tom, Obj2
John, Obj2
John, Obj1
Lina, Obj1
Lina, Obj2

Partition set w.r.t. read

Ray, Obj1
Tom, Obj1
Ray, Obj2
Tom, Obj2
John, Obj2
John, Obj1
Lina, Obj1
Lina, Obj2

Partition set is conflict-free w.r.t. read and write → YES

@Shuvra Chakraborty
Given an operation \( op \), if partition set is conflict-free and each partition is uniquely identified by the set of (attribute name, value) pair then RuleSet can be generated [Proved].

A conjunction of (attribute name, value) pair is made for each conflict-free bold black partition and OR’ed to Rule\(_{\text{op}}\).

\[ \text{e.g., Rule}_{\text{read}} \equiv <(u\text{roleAtt}(u) = \{R3\} \land o\text{roleAtt}_{\text{write}}(o)=\{R1\} \land o\text{roleAtt}_{\text{read}}(o)=\{R1, R3\}) \lor (u\text{roleAtt}(u) = \{R1, R2, R3\} \land o\text{roleAtt}_{\text{write}}(o)=\{R1\} \land o\text{roleAtt}_{\text{read}}(o)=\{R1, R3\})> \]

***Rule\(_{\text{write}}\) can be constructed same way

*RuleSet = \{Rule\(_{\text{write}}, \text{Rule}_{\text{read}}\}\*

Example 2 and completed ABAC system in example 3 are equivalent

***Equivalent ABAC system generation is always possible!
Role Based Access Control System (Ex. 2)

Equivalent ABAC system

(b) With supporting data

Supporting Data (Ex. 3)

<table>
<thead>
<tr>
<th>UAValue</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>User (U)</td>
<td>Position</td>
<td>Dept.</td>
</tr>
<tr>
<td>John</td>
<td>Officer</td>
<td>CS</td>
</tr>
<tr>
<td>Lina</td>
<td>Student</td>
<td>CS</td>
</tr>
<tr>
<td>Ray</td>
<td>Officer</td>
<td>CS</td>
</tr>
<tr>
<td>Tom</td>
<td>Officer</td>
<td>CS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OAValue</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Object (O)</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>Obj1</td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>Obj2</td>
<td>Printer</td>
<td></td>
</tr>
</tbody>
</table>

RangeSet

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>{Officer, Student, Faculty}</td>
</tr>
<tr>
<td>Dept.</td>
<td>{CS, EE}</td>
</tr>
<tr>
<td>Type</td>
<td>{File, Printer, Scanner}</td>
</tr>
</tbody>
</table>

Step 1: Generate partition set based on similarity in attribute value assignment. Partition set might have conflicts!

@Shuvra Chakraborty
*Partition set has conflict w.r.t. write → YES (Ex. 2 and 3)
Next step: Apply infeasibility correction
Partition the sets of users and objects present. For example, \{John, Ray, Tom\} is partitioned as \{John\} and \{Ray, Tom\}

Partitioning:
- **Conflict-free** \((UA, OA)\)

Generate a conjunctive clause:
- Conflict-free \((UA \cup u\text{roleAtt}), (OA \cup o\text{roleAtt}_o, op\in OP)\)

**If needed**:
- a. Add role-based user attribute to \(UA\)
- b. Add role-based object attributes to \(OA\)

Use role-based attribute values to identify:
- \{John\} and \{Ray, Tom\} are assigned different role-based attribute values

**Infeasibility correction**: exact solution can be achieved many ways.

@Shuvra Chakraborty
\[\text{Rule}_{\text{write}} \equiv \langle \text{Position}(u) = \text{officer} \land \text{Dept}(u) = \text{CS} \land \text{uroleAtt}(u) = \{\text{R1, R2, R3}\} \land \text{Type}(o) = \text{File} \rangle \lor \langle \text{Position}(u) = \text{officer} \land \text{Dept}(u) = \text{CS} \land \text{uroleAtt}(u) = \{\text{R1, R2, R3}\} \land \text{Type}(o) = \text{Printer} \rangle \lor \langle \text{Position}(u) = \text{student} \land \text{Dept}(u) = \text{CS} \land \text{Type}(o) = \text{Printer} \rangle> \]

*\text{RuleSet} = \{\text{Rule}_{\text{write}}, \text{Rule}_{\text{read}}\}
Formalized notion of feasibility on RBAC to ABAC policy mining: first time

The overall asymptotic complexity of ABAC RuleSet Existence problem is $O(|OP| \times (|U| \times |O|))$

The overall asymptotic complexity of ABAC RuleSet Infeasibility Correction in RBAC context is $O(|OP| \times (|U| \times |O|)^3)$

Challenges

- Can you ensure partition split always equals 2?
- More compact set of rule generation
- Negative rules?
- Exact solution:
  - Reduce number of split partitions
  - Change number of attributes required
  - Changing existing attribute set, possible?