Synthesizing and Analyzing Attribute-Based Access Control Model Generated from Natural Language Policy Statements

Mahmoud Abdelgawad  Indrakshi Ray  Saja Alqurashi  Videep Venkatesha

Computer Science Department, Colorado State University

Hossein Shirazi
Management Information Systems, San Diego State University

SACMAT June 7-9, 2023
Motivation

• Security requirements are written in natural language
• Access requirements are manually extracted from these documents, represented by some access control model, and then enforced by security mechanisms
• Manual extraction can be tedious and error-prone
• Access control errors can have grave consequences
• *Can we automatically generate access control models from security requirements?*
Access Control Model

- We focus on NIST Next Generation Access Control Model
- Supports attribute-based access control
  - Entities: users, user attributes, objects, object attributes, policy classes
  - Relations: assignments, associations, prohibition, obligation
- Dynamic access control model: can change policies while they are deployed
- Example usage: situation monitoring applications, applications spanning multiple domains
NGAC Security Model Example

Human Resource File System
• 3 Users
• 2 Resources
• 3 User Attributes
• 2 Resource Attributes
• 1 Policy Class
• 9 Assignments
• 2 Association

AR= \{\text{read, write}\}
SpaCy Model: Linguistic Features

- **Tokenization**: splits texts into words, punctuation marks etc. (tokens)
- **Part-of-speech (POS) Tagging**: labels tokens as noun, verb, adjective etc.
- **Lemmatization**: assigns the base forms of the word
- **Dependency Parsing**: parses the tokens, generates a syntactic dependency tree, and assigns relations to the links
- **Named Entity Recognition (NER)**: identifies various named and numeric entities
SpaCy Model Cont.

- Example: “The administrator can choose a hospital.”

Syntactic Dependency Tree
NGAC Security Model Extraction

**Extraction Framework:**
1) Apply NLP to each ACP statement
2) Define the relation type
3) Extract the elements and form them into NGAC relation
4) Present the elements and the relations into Neo4j as graph
NGAC Security Model Extraction
Cross-functional Flowchart
NGAC Extraction: Assignment Rules

- Proper noun subject converted to user
- Common noun subject converted to user attribute
- Proper noun object converted to resource
- Common noun object converted to resource attribute
- User to user attribute relation created
- Resource to resource created
NGAC Extraction: Prohibition Rules

• Syntactic dependency contains a negation pattern
  • a user cannot view, or a user is not allowed to view an object)
• Root verb is checked against a list of prohibition words
  • prohibit, deny, disallow, forbid, prevent, inhibit

HCP is prevented to view the patient’s security question and password

\[ Prohibition(deny(HCP - \{view\} - security\ question)) \]
\[ Prohibition(deny(HCP - \{view\} - password)) \]
NGAC Extraction: Obligation Rules

- Syntactic dependency has an adverbial clause as (event and response)
- Independent clause represents the response
- Conditional clause corresponds to the event

Whenever the HCP changes the patients’ data, an email must be sent to the administrator

\[
\text{Obligation(} \text{Event : (HCP – \{change\} – patient record)} \rightarrow \text{Response : (send(email) – administrator)})
\]
NGAC Extraction: Association Rules

• If the ACP sentence is not obligation or prohibition, is it association?
• Syntactic dependency is used to identify association relation
  • (subject, predicate, object)

The administrator can choose a hospital

Association(administrator – {choose} – hospital)
Represent NGAC Elements & Relations in Neo4j

1. Association Statement (UA-[AR]-RA):
   Doctor can update the patient record

2. User Assignment (U-UA):
   Bob is a doctor

3. Resource Assignment (R-RA):
   John Smith Record is a patient record
Represent NGAC Elements & Relations in Neo4j

**iTrust: Access Control Policies**

1. An HCP creates patients.
2. Doctor is an HCP.
3. Nurse is an HCP.
4. A doctor is prohibited from creating patients.
5. Doctors can update the patient record.
6. Nurses can only view the patient record.
7. Whenever a HCP changes a patient record, an email must be sent to the administrator.
8. The administrator can assign a patient.

**Users**

9. Bob is a doctor.
10. Alice is a nurse.
11. Jack is an administrator.

**Resources**

12. JohnSmithRecord is a patient record.
13. John is a patient.

8 ACP Sentences from iTrust Dataset + 3 Users and 2 Resources

NGAC Graph

SACMAT June 7-9, 2023 6/7/23 14
Generate Paths from Neo4j Graph

- Use Neo4j Cypher to generate paths
- Each path represents access of user to resource
- 8 Paths that cover the graph from Users to Resources.

Example of two paths:

1) [{name:"Bob"}, {assign:"assigned_to"}, {name:"doctor"}, {name:"doctor"}, {ars:"update"}, {name:"patient record"}, {name:"patient record"}, {assign:"assigned_to"}, {name:"John Smith Record"}].

2) [{name:"Alice"}, {assign:"assigned_to"}, {name:"nurse"}, {name:"nurse"}, {ars:"view"}, {name:"patient record"}, {name:"patient record"}, {assign:"assigned_to"}, {name:"John Smith Record"}].
Testing Process

- Test Scripter loads the generated paths and converts them into testcases
- Test Executor runs the testcases against each ACP, reports the result (pass or fail)
- Applies to 7 Datasets collected from the Web
- Verify the well-formedness properties: Completeness, Consistency, and Minimality
Testing Process Cont.

- **Use Regular Expression to Script Testcases**
  - e.g., ACP Sentences:
    1) HCP can create patient
    2) Bob is HCP
    3) John Smith is a patient

Path:
```
[{'name':"Bob"},{'assign':"assigned_to"},{'name':"HCP"},{'name':"HCP"},{'ars':"create"},{'name':"patient"},{'name':"patient"},
{'assign':"assigned_to"},{'name':"John"}]
```

Regular Expression Pattern = "[{name:"Bob"}.*{name:"HCP"}.*{ars:"create"}.*{name:"patient"}.*{name:"John Smith"}]"
Testing Process Cont.

- **Completeness:** Awkward sentences, unnecessary words, incorrect statements

<table>
<thead>
<tr>
<th>Institute</th>
<th>Dataset Name</th>
<th># of ACPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Massachusetts Amherst</td>
<td>MA</td>
<td>2</td>
</tr>
<tr>
<td>University of Arizona</td>
<td>AZ</td>
<td>9</td>
</tr>
<tr>
<td>Georgia State University</td>
<td>GA</td>
<td>20</td>
</tr>
<tr>
<td>Colorado State University</td>
<td>CSU</td>
<td>62</td>
</tr>
<tr>
<td>New York University</td>
<td>NYU</td>
<td>83</td>
</tr>
<tr>
<td>University of Denver</td>
<td>DU</td>
<td>105</td>
</tr>
<tr>
<td>Harvard University</td>
<td>Harvard</td>
<td>249</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of TCs</th>
<th># of TCs Passed</th>
<th>Accuracy(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>16</td>
<td>14</td>
<td>87.50</td>
</tr>
<tr>
<td>AZ</td>
<td>31</td>
<td>23</td>
<td>74.19</td>
</tr>
<tr>
<td>GA</td>
<td>248</td>
<td>234</td>
<td>94.35</td>
</tr>
<tr>
<td>CSU</td>
<td>162</td>
<td>146</td>
<td>90.12</td>
</tr>
<tr>
<td>NYU</td>
<td>271</td>
<td>215</td>
<td>79.34</td>
</tr>
<tr>
<td>DU</td>
<td>440</td>
<td>388</td>
<td>88.18</td>
</tr>
<tr>
<td>Harvard</td>
<td>509</td>
<td>359</td>
<td>70.53</td>
</tr>
</tbody>
</table>

Avg. 83.5%
Testing Process for Consistency

- **Consistency** (Paths-conflict): grant and deny a user for the same resource
  - HCP can create a patient
  - Doctor cannot create a patient
  - Bob is a doctor and HCP.
  - John Smith is a patient.

Two paths conflict:
1) ```
   [{name:"Bob"}, {assign:"assigned_to"}, {name:"doctor"}, {name:"doctor"}, {Deny:"create"}, {name:"patient"},
   {name:"patient"}, {assign:"assigned_to"}, {name:"John Smith"}].
```

2) ```
   [{name:"Bob"}, {assign:"assigned_to"}, {name:"HCP"}, {name:"HCP"}, {ars:"create"}, {name:"patient"}, {name:"patient"},
   {assign:"assigned_to"}, {name:"John Smith"}].
```
Testing Process for Minimality

- **Minimality** (Non-redundancy): repeated access right for the same user in various ACP
  - HCP can create a patient
  - Bob is HCP.
  - Bob can create a patient.
  - John Smith is a patient.

Two paths are redundant:

1. `[{name:"Bob"},{assign:"assigned_to"},{name:"HCP"},{name:"HCP"},{ars:"create"},{name:"patient"},{name:"patient"}, {assign:"assigned_to"},{name:"John Smith"}]`. 

2. `[{name:"Bob"},{assign:"assigned_to"},{name:"HCP"},{name:"HCP"},{ars:"create"},{name:"patient"},{name:"patient"}, {assign:"assigned_to"},{name:"John Smith"}]`. 

Scalability: Time and Space Complexity

Quadratic Time Complexity

Only Assignment Relations Reached 1200 Nodes for 249 ACP Statements
Future Work

- Clean ACP Datasets
- Multiple levels of hierarchy
- Use graph properties for efficient verification
- Comparison to other NLP models BERT and GPT-3
- Use formal methods to verify whether use case and misuse case scenarios generated from software and security specification complies with the policies