DynPolAC: Dynamic Policy-based Access Control for IoT Systems

Mehdi Karimibiuki, Ekta Aggarwal, Karthik Pattabiraman and Andre Ivanov

December 6, 2018
The 23rd IEEE Pacific Rim International Symposium on Dependable Computing
Taipei, Taiwan
Motiation: IoT Space

- The number of IoT systems are growing
- About 26 devices per person
Motivation: mobile IoT by 2020

- 7 million unmanned aircraft systems (UAS)
- 10 million connected vehicles
- 1 in 4 cars are autonomous by 2030
Autonomous IoT: Drones

(1) Moving objects have higher levels of interaction than stationary networks

(2) with linear growth of IoT nodes, communication between them grow quadratically $\frac{n(n-1)}{2}$
Problem: Malicious Attacks

Our Goal: Develop an authorization scheme for highly interactive IoT systems
Challenges

Dynamic IoT nodes are constrained systems
- Weight limitation
- Power consumption

High interaction means fast authorization required

Communication in congested networks could build up quadratically → so does authorization
Previous Work

XACML Policy-based

- Used in cloud or big data context
- Speed does not matter; not doing the parsing often.

Classical Models
RBAC, CBAC, TBAC

- Ad-hoc models mainly depends on the platform; not standalone

[Kim, Fysarakis, Seitz, Turkmen]

[Sandhu, Rigazzi, Mahalle]
Our Goal

No authorization scheme exists that can support data access in both fast and expressive way.

- XACML Policy-based
- DynPolAC
  - Expressive for IoT environments
  - 4 x faster
- Classical Models
  - RBAC, CBAC, TBAC

[Kim, Fysarakis, Seitz, Turkmen]

[Sandhu, Rigazzi, Mahalle]
Outline

• Motivation
• Approach
• evaluation
DynPolAC: Key Insight

We describe rules in high level language
- Reduce syntax size
- Save the parsing time

Use only the necessary expressions required in IoT space

remove unnecessary nested elements and make simple syntax

- Will show even in small embedded platforms DynPolAC is fast and meets the overall speedup in the system performance.
DynPolAC: No-fly Zone

We can construct rules with 6 primitives only

1. Access type: Permit
2. Data type: coordinates
3. Drone name: Friendly
4. Time: ALL
5. User: UTM
6. Group: Airport

- Radar sees an unknown asset
- UTM starts communicating with the drone to re-route

➤ Spatial, temporal, and role-based expressions can be built with 6-element policy blocks
DynPolAC: Comparison

Let’s see how rules look in previous model?

DynPolAC

1. <policy>
2.  <rule>access</rule>
3.  <attributes>
4.   <type>email</type>
5.   <vendor>MediCorp</vendor>
6.   <time>ANY</time>
7.   <user>ANY</user>
8.   <group>med.example.com</group>
9.  </attributes>
10. </policy>

removed unnecessary nested elements, still 6 primitives, made simple syntax

10 vs. 39

Outline

• Motivation
• Approach
• evaluation
Research Questions

• RQ1. At micro-level, what is the processing time improvement?

• RQ2. At system-level, what is the response time?
  ➢ Check stability condition
    o Can it meet requests in interactive environments?
  ➢ Sensitivity analysis
    o What is the bottleneck in extreme scenarios?

• RQ3. What is the memory overhead?
Experimental Setup

Instrument DynPolAC in 3 different embedded platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi Zero (Pi0)</td>
<td>1200</td>
</tr>
<tr>
<td>Raspberry Pi Three (Pi3)</td>
<td>1000</td>
</tr>
<tr>
<td>Beagle Bone Black (BBB)</td>
<td>720</td>
</tr>
</tbody>
</table>

Goal: show the homogeneity of our results in different platforms
Experimental Setup

System Study: Emulate an interactive IoT environment

Discrete event simulator

Objects Population

Response Time

Policy check

Node Database

FCFS
RQ1. Processing Time

Rules by DynPolAC syntax are parsed and processed in milliseconds. Less than half seconds.
RQ1. Comparison

- On Average 4x process improvements
- Up to 7.27x speedup
- Speedup is higher in slower platforms
- DynPolAC is a suitable scheme for low-capacity devices

✅ On Average 4x process improvements
✅ Up to 7.27x speedup
RQ2. System Stability Condition

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival rate ($\lambda$)</td>
<td>1/s</td>
<td>1 - 8</td>
</tr>
<tr>
<td>Size of query</td>
<td>Bytes</td>
<td>200 – 5K</td>
</tr>
<tr>
<td>Size of policy</td>
<td>No. of rules</td>
<td>1 - 2000</td>
</tr>
</tbody>
</table>

Goal: measure system performance by calculating the response time

The end-to-end time of a drone to initiate the request until the reply is received.
RQ2. System Stability Condition

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Arrival rate ($\lambda$)</td>
<td>1/s</td>
<td>4</td>
</tr>
<tr>
<td>Size of query</td>
<td>Bytes</td>
<td>200 – 5K</td>
</tr>
<tr>
<td>Size of policy</td>
<td>No. of rules</td>
<td>1 - 2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>simulation</th>
<th>response time (ms)</th>
<th>Response rate ($\mu$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-policy</td>
<td>178</td>
<td>5.6</td>
</tr>
<tr>
<td>DynPoIAC</td>
<td>245</td>
<td>4.1</td>
</tr>
<tr>
<td>XACML</td>
<td>840</td>
<td>1.2</td>
</tr>
</tbody>
</table>

DynPoIAC satisfies the stability condition being right above the threshold of 4.
RQ2. Stability Condition

<table>
<thead>
<tr>
<th>simulation</th>
<th>response time (ms)</th>
<th>Response rate (µ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-policy</td>
<td>178</td>
<td>5.6</td>
</tr>
<tr>
<td>DynPolAC</td>
<td>245</td>
<td>4.1</td>
</tr>
<tr>
<td>XACML</td>
<td>840</td>
<td>1.2</td>
</tr>
</tbody>
</table>

XACML-based policy systems experience instability, so it cannot keep up requests!!

DynPolAC improves the overall response time by 70%.
RQ2. Sensitivity Study

**Extreme case**

- Arrival Rate: 8/s
- Data size: 2kB

![Graph showing Time (ms) vs. Number of Rules for DynPolAC and XACML-based methods]

Sweep the number of rules
RQ3. Memory Overhead

DynPolAC incurs only 7.5% memory overhead compared to the rest of our system.

Can be deployed to memory constrained nodes

Summary

• Looked at a scenario of dynamic IoT system
  • DynPolAC is the solution to securely authenticate dynamic objects
• Insight: DynPolAC has a crisp language selection
  • high-level language to express very low-level parameters.
  • expresses similar rules compared to previous work.
  • Suitable for constrained IoT nodes with only 7.5% overhead.
  • Up to 7.28x speedup achieved, 4x on average.

• DynPolAC guarantees system stability.

Mehdi Karimi; Email: mkarimib@ece.ubc.ca