JSON-encoded ABAC (XACML) policies

Steven Davy, Bernard Butler, Leigh Griffin, Brendan Jennings

FAME project
Waterford Institute of Technology

Presentation to OASIS XACML TC concerning JSON-encoded XACML policies, 2013-05-30
Outline

1. Introduction
   Motivation
   Response to challenge
   Our contribution

2. Evaluation
   Setup
   Results

3. Discussion and Conclusions
Architecture outline

Source: XACML 2.0 specification
Problems with XACML’s XML encoding

Policies

- **Verbose**
  - difficult to edit manually
  - communication and processing overhead
- **Not “executable”**
  - Harder to build user-friendly tooling c.f. Java IDEs
  - PDP has more work to do: functions, combination algorithms, . . .

Requests (and responses)

- Modern web services use “lighter” request/response formats
- Potential performance bottleneck - format conversion
## Industry response

<table>
<thead>
<tr>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Improved policy authoring tools (e.g., Quest One APS, Cisco EPM)</td>
</tr>
<tr>
<td>• Transforming simpler language to XACML (e.g., ALFA from Axiomatics)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>OASIS XACML 3.0 JSON requests and responses!</td>
</tr>
</tbody>
</table>
### Outstanding problems: now

#### Policies
- Greater complexity of policy authoring toolchain
- Continued limited support for policy analysis
- PDP still has to handle multiple concerns
- Lost opportunity to apply *modern principles*

#### Requests
- OASIS activity has great potential to reduce impedance mismatch with clients
- Potential complexity if encoding for policies and for requests differ
Our contribution: language

<table>
<thead>
<tr>
<th>JSON policies and “context”</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Developed independently of OASIS XACML TC</td>
</tr>
<tr>
<td>• Developed analog of XACML 2.0: JSONPL\textsuperscript{a}</td>
</tr>
<tr>
<td>• Subject, Resource, Action, Environment attributes</td>
</tr>
<tr>
<td>• Rules and rule/policy combination</td>
</tr>
<tr>
<td>• 4-valued decisions: Permit, Deny, Indeterminate, Not Applicable</td>
</tr>
<tr>
<td>• JSONPL currently lacks</td>
</tr>
<tr>
<td>– Obligations</td>
</tr>
<tr>
<td>– XACML 3.0 new features (delegation, etc.)</td>
</tr>
<tr>
<td>– Formal “schema”</td>
</tr>
</tbody>
</table>

\textsuperscript{a}JSON Privilege Language
Language features and tools

Policy editing

- Structure of policy is identical to XACML 2.0
  - keywords and much of the syntax
  - hierarchies
  - relationships (grouping)
- Language features
  - (nested) associated arrays (hashes) and enumerated arrays
  - High-level keys: XACML 2.0 elements (Subject, Rule, etc.)
  - Low-level keys: Specific subject attributes, etc.
- Remove type declarations (string, number, date only)
- Remove namespace handling
- Use text editor; many are JSON-aware (well-formed)
Example JSONPL policy (fragment)

"Policy":{
  "id":"RPSlist.7.0.1",
  "target":{
    "subjects":{
      "subject":{
        "role":"admin"
      }
    },
    "resources":{
      "resource":{
        "isPending":"false"
      }
    },
    "actions":{
      "action":{
        "action-type":"write"
      }
    }
  }
},
"rule":{
  "id":"RPSlist.7.0.1.r.1",
  "effect":"permit"
}
}

Overall XACML policy set: 420Kb (311Kb).
Overall JSONPL policy set: 70Kb (17Kb).
Example JSONPL request (manually-generated)

```json
{
    "subject": {
        "category": "access-subject",
        "role": "pc-chair"
    },
    "resource": {
        "isPending": "false",
        "resource-id": "DEFAULT RESOURCE"
    },
    "action": {
        "action-type": "write"
    }
}
```

Size comparison

- XACML request: 872b (842b).
- JSONPL request: 211b (158b).
Our contribution: prototype

Node.js, JavaScript, redis

- Node.js: server side JavaScript framework: new wave of scalable web services
- JavaScript: Reduced friction—JSONPL is based on JavaScript syntax
- redis: NoSQL, highly performant, key-value store—very fast lookup by key
- Present components: PDP, PEP, PRP
  - Prototype code!!!
  - Limited scope—see exclusions above
  - Limited robustness—edge cases, invalid input, etc.
Rationale for design decisions

Trends in web service development and deployment

- Traditional: Java/.NET, XML, SOAP
- Newer: Java/Scala/JavaScript/..., JSON, REST
- Growth of functional and domain-specific languages

Analysis of requirements

- Less verbose, but just as expressive
- (Semi-)executable (in JavaScript-based PDP)
- Low friction (from client request to server policy and back)
- Better performance (described in POLICY 2012 paper)
## Setup

### Policies and Requests
- CONTINUE-A policy set\(^a\) (60 policies approx)
- Example requests (200 requests approx)
- Verification: Same decisions as *SunXACML* PDP

XACML $\rightarrow$ JSONPL conversion: done manually by Leigh
For comparison: automated XML-to-JSON

\(^a\)Distributed with Xengine PDP

### PDP, PRP, PEP
- PDP: Match JSON elements, lookup rule value, combine as necessary
- PEP: very light, forwards requests and responses
- PRP: thin wrapper for redis policy store
Performance experiments

STACS (Scalability Testbed for Access Control Systems)

- Factorial experiments: server, PDP, conversion_type, ...
- Run tests, collect service times, etc.
- Analyze, compare and visualize results
## Summary of Results

### JSONPL
- ABAC with XACML “metamodel” and JSON encoding: proof of concept
- Reduced verbosity without reducing expressiveness
- Minimal friction (and even less with JavaScript-based PDP)

### Prototype
- JavaScript PEP, PDP, PRP with JSON Privilege Language: proof of concept
- Simplicity of JavaScript-based PDP with JavaScript-based policies and context
- Performance and resource usage: dramatic improvements over *SunXACML* PDP
- Quantified improvements: conciseness $\ll$ language choice
Performance improvements

- SX is *SunXACML* PDP, njsr is the Node.js-redis prototype PDP.
- bear and inisherk are server instances.
Strengths and weaknesses

- Results are very encouraging, but need to . . .
  - consider larger/more complex policy sets
  - policy editor/transformation tools
  - reference implementation, like SunXACML
  - consider migration path for existing XACML users and vendors
  - encourage STANDARDS and INDUSTRY ADOPTION!
Conclusions

Summary

• Presentation triggered by discussion on xacml-users list
• We welcome the opportunity to develop this further

Selected bibliography


Thanks for your attention!