Authorization Recycling in RBAC Systems

Qiang Wei¹, Jason Crampton², Konstantin Beznosov¹, Matei Ripeanu¹

¹Laboratory for Education and Research in Secure Systems Engineering (LERSSE), University of British Columbia
²Information Security Group, Royal Holloway, University of London
outline

- the overview
  - authorization architecture
  - motivation
  - recycling approach
- recycling algorithms
- experimental evaluations
- summary & future work
a typical authorization architecture

- also known as request-response paradigm
- applied by IBM Access Manager, Entrust GetAccess, CA SiteMinder, etc.
motivations

+ re-use of authorization logic
+ consistent policy enforcement

- reduced availability
- increased latency
- reduced scalability
existing approaches

- fault-tolerance by replication/redundancy
  - + improve availability
  - - latency remains unchanged
  - - require specialized OS/middleware
  - - poorly scale on large populations

- caching previous authorizations
  - + simple, inexpensive
  - + improves performance & availability
  - - serves only requests that have been issued before (precise recycling)
The Secondary and Approximate Authorization Model (SAAM)\(^1\)

1. resolve returning requests (precise recycling)
2. resolve new requests (approximate recycling)

---

SAAM<sub>RBAC</sub>

- **SAAM**
  - only an abstract model
  - a specific SAAM recycling algorithm is needed for each access control model

- **SAAM<sub>RBAC</sub>**
  - apply SAAM to role-based access control (RBAC) model
  - develop recycling algorithms specifically for RBAC
outline

- the overview
- recycling algorithms
- experimental evaluations
- summary & future work
terminology

- request: issued by a subject $s$ for a permission $p$
  - $\text{request}= (s, p)$
- $\pm$: denotes the decision to a request
  - an allow response: $+(s, p)$
  - a deny response: $-(s, p)$
- subject: modeled as the set of roles $r$ activated in a session
  - $s = \{r_1, r_2, r_3\}$
inference rules

- **Rule\(^+\):** if \(+(s,p)\) and \(s' \subseteq s\), then request \((s',p)\) should also be **allowed**

- **Rule\(^-\):** if \(-(s,p)\) and \(s' \subseteq s\), then request \((s',p)\) should also be **denied**

\[
S_1 = \{r_1\} \quad \subseteq \\
S_2 = \{r_1, r_2\} \\
\text{Rule}^+ \quad \text{Rule}^- \\
\]
SAAM$_{RBAC}$ recycling algorithms

- cache construction
- decision
- cache update

PEP

SDP

PDP

Primary responses

Secondary responses

Policy change messages
cache$^+$ and cache$^-$

SDP

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>${r_3, r_4}, {r_5, r_6}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_2$</td>
<td>${r_1}, {r_5, r_6}$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>${r_2, r_4}, {r_1, r_2}$</td>
</tr>
</tbody>
</table>

Each $p_i$ is built from allow responses in $\text{cache}^+$ and from deny responses in $\text{cache}^-$.

PDP

built from allow responses

built from deny responses
example: SDP receives 1\textsuperscript{st} deny response

- (\{r_1, r_2\}, p)
example: SDP receives 1\textsuperscript{st} allow response

\[ + \{r_2, r_3, r_4\}, p \]

\[ \{r_1, r_2\} \]

SDP
example: SDP receives 2\textsuperscript{nd} allow response

SDP

\begin{align*}
\text{cache}^+ & \rightarrow \{\{r_3, r_4\}\}\{r_4, r_5, r_6\} \\
\text{cache}^- & \{r_1, r_2\}
\end{align*}

+ \{r_4, r_5, r_6\}, p

PDP
example: SDP receives 2\textsuperscript{nd} deny response

SDP

\begin{align*}
\text{cache}^+ &: \{(r_3, r_4), (r_4, r_5, r_6)\} \\
\text{cache}^- &: \{(r_1, r_2), (r_4, r_7)\}
\end{align*}

PDP

\[-({r_4, r_7}, p)\]
example: SDP receives 2\textsuperscript{nd} deny response

\begin{itemize}
  \item cache\textsuperscript{+}
    \begin{align*}
    \{ \{ r_3 \}, \\
    \{ r_5, r_6 \} \}
    \end{align*}
  \item cache\textsuperscript{-}
    \begin{align*}
    \{ r_1, r_2, \\
    r_4, r_7 \}
    \end{align*}
  \end{itemize}

- (\{ r_4, r_7 \}, p)
example: SDP makes an allow decision

$$(\{r_3, r_4\}, p)$$

PEP allows

SDP: $\text{cache}^+ = \{r_3\}$, $\{r_5, r_6\}$

SDP: $\text{cache}^- = \{r_1, r_2, r_4, r_7\}$
example: SDP makes a deny decision

$((\{r_1, r_4, r_7\}, p))$
example: SDP makes an undecided decision

PEP

undecided

SDP

({r_1, r_5}, p)

(cache^+, \{\{r_3\}, \{r_5, r_6\}\})

(cache^-, \{r_1, r_2, r_4, r_7\})
**example: summary**

- algorithm correctness is proved
  - if the SDP makes any allow or deny decision, the PDP will always make the same decision
Outline

- the overview
- recycling algorithms
- experimental evaluations
- summary and future work
evaluation metrics

- SDP hit rate
  - a cache hit
    - a request is resolved by the SDP
  - higher hit rate => more requests resolved by the SDP
    - even when the PDP fails => higher availability
    - reducing the load of the PDP => higher scalability

- SDP inference time
  - the time used to infer approximate responses
  - less inference time, more efficient the system
evaluation methodology

- input parameters
- evaluation engine
- data input files
  - warming request set
  - testing request set
  - RBAC policy
- PDP
- cache^+
- cache^-
- SDP

warming phase

testing phase
compared with simple caching, hit rate is improved significantly by using $\text{SAAM}_{\text{RBAC}}$ recycling algorithm
the impact of various system parameters

- the percentage of deny responses
- the number of roles
- the number of roles assigned per permission
- the number of roles assigned per user
- the popularity distribution of role assignment
- ...
inference time

RABC policy: 100 subjects, 1,000 objects, 50 roles

inference time stabilizes
cache size

RABC policy: 100 subjects, 1,000 objects, 50 roles

cache size stabilizes
outline

- the overview
- recycling algorithms
- experimental evaluations
- summary and future work
summary

- re-use of authorization logic
- consistent policy enforcement
- lower admin overhead

- reduced availability
- increased latency
- reduced scalability

SDP

data input files
- warming request set
- testing request set
- RBAC policy

input parameters
- precise recycling component
- approximate recycling component

evaluation engine
- warming
- testing

PDP

primary responses
policy change message

cache construction
decision
cache update

hit rate (%)
cache warmness (%)
future work

- when role hierarchy is available
- cache replacement algorithm
- experiment with real enterprise RBAC policies and request traces
We are looking for policies and traces from real applications! If you are willing to share them, please talk to me or contact me at: qiangw@ece.ubc.ca