ACaaS: Access Control as a Service for IaaS Cloud

Ruoyu Wu*, Xinwen Zhang†, Gail-Joon Ahn*, Hadi Sharifi*, Haiyong Xie†
*Arizona State University
†Huawei Research Center
Outline

- **Introduction**
  - Motivation Scenarios
  - ACaaS Overview
  - AWS IAM and Its Limitations

- **ACaaS\(_{RBAC}\) for AWS**
  - Why RBAC
  - Challenges of Supporting RBAC for AWS
  - ACaaS\(_{RBAC}\) model

- **System Design**

- **Implementation and Evaluation**

- **Related Work**

- **Conclusion and Future Work**
Motivation Scenarios

- Cloud Sandbox for Software Development

- Government cloud platform
  - Regulation compliance requirement (e.g., HIPAA, SOX)
  - Multi-level security requirement
ACaaS Overview

- The diversity of cloud services across a wide range of orgs and domains requires various security models and policies.
- However, major cloud providers all fail to accommodate this.
- Hence, we need a comprehensive and adaptive access control mechanism in cloud.

Diagram:

- OS Core Services: RBAC, MLS, DTE, Process Mgmt, I/O Operations, Memory Mgmt, File System, DBMS
- Security
- Hardware Resources
- Operating System
- ACaaS Approach: Amazon Web Services, Windows Azure
- Personal Cloud App, Enterprise Cloud App, Government Cloud App
- Access Control as a Service (ACaaS): MAC, CW, RBAC, Compute, Networking, Content Delivery, Payment, Management, Database

**AWS IAM Service**

- **IAM model**
  - Users, Groups, Action, Objects, Permissions, Constraints, UGA, PUA and PGA.

- **Policy**
  - 4-tuple P = (user, permission, constraint, effect)

```json
{"Statement": ["Effect":"Allow", "Action": ["iam:CreateAccessKey", "iam:ListAccessKeys"], "Resource": "arn:aws:iam::123456789012:user/**", "Condition": { "DateGreaterThan": { "aws:CurrentTime": "2010-07-01T00:00Z" }}, ]}
```
AWS IAM Service - Cont'd

- Limitations
  - A very fine-grained approach
  - Groups are organized in a flat structure
  - Only static constraints, SoD is missing
  - No session management
  - Does not separate administrators and regular users clearly

- IAM Role
  - It only supports assigning IAM roles to an EC2 instance level but not to user level
  - An IAM role does not support session management
  - EC2 role does not support other important RBAC features such as role hierarchy and delegation
In this work, we present ACaaS\textsubscript{RBAC}, a reference architecture of ACaaS that supports RBAC for Amazon AWS cloud platform.

Why RBAC

- It has been widely adopted in enterprise applications
- It is a very generic access control model
- It can address all the identified limitations of AWS IAM service.
ACaaS_{RBAC} for AWS - Cont'd

- Challenges to support RBAC for AWS
  - Efficient role hierarchy management
  - Session management
  - SoD support and management of privileged account
  - System integration and minimal overhead

- ACaaS_{RBAC} addresses these challenges with a service-oriented RBAC for AWS cloud resources
ACaaS$_{\text{RBAC}}$ Model

- Beyond the existing RBAC96 and NIST RBAC model, ACaaS$_{\text{RBAC}}$ integrate the *Group* concept of IAM into RBAC
  - Group can be a good counterpart to role for better managing user sets and relatively static permission sets
ACaaS_{RBAC} Model - Cont'd

- **Core Model**
  - Sets: $U, AU, G, R, AR, P, AP, S$
  - Relations: $UR, PR, RH, UG, PG, AUAR, APAR$
  - Functions: $Sessions: U \rightarrow 2^S; Roles: S \rightarrow 2^R; Permissions: S \rightarrow 2P$

- **Role-based Administrative Model**
  - Admin resource $RES_A \subseteq U \cup G \cup P \cup R$
  - Admin action $ACT_A = \{Create, Delete, Assign, Revoke\}$
  - Admin scope $SCO \subseteq 2^U \times 2^G \times 2^P \times 2^R$
  - Admin policy 4-tuple $p_a = <ar, act, res_a, aco>$, where $ar \in AR, act \in ACT_A, res_a \in Get\_RES_A(sco)$, and $sco \in Scopes(ar)$
### ACaaS$_{RBAC}$ Model - Cont'd

- **Administrative action table**

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Postconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create(u)</td>
<td>N/A</td>
<td>$u \in U, u \in \text{Scope}_U(\text{ar})$</td>
</tr>
<tr>
<td>Delete(u)</td>
<td>$u \in \text{Scope}_U(\text{ar})$</td>
<td>$u \notin U, u \notin \text{Scope}_U(\text{ar})$</td>
</tr>
<tr>
<td>Create(g)</td>
<td>N/A</td>
<td>$g \in G, g \in \text{Scope}_G(\text{ar})$</td>
</tr>
<tr>
<td>Delete(g)</td>
<td>$g \in \text{Scope}_G(\text{ar})$</td>
<td>$g \notin G, g \notin \text{Scope}_G(\text{ar})$</td>
</tr>
<tr>
<td>Create(p)</td>
<td>N/A</td>
<td>$p \in P, p \in \text{Scope}_P(\text{ar})$</td>
</tr>
<tr>
<td>Delete(p)</td>
<td>$p \in \text{Scope}_P(\text{ar})$</td>
<td>$p \notin P, p \notin \text{Scope}_P(\text{ar})$</td>
</tr>
<tr>
<td>Create(r)</td>
<td>N/A</td>
<td>$r \in R, r \in \text{Scope}_R(\text{ar})$</td>
</tr>
<tr>
<td>Delete(r)</td>
<td>$r \in \text{Scope}_R(\text{ar})$</td>
<td>$r \notin R, r \notin \text{Scope}_R(\text{ar})$</td>
</tr>
<tr>
<td>Assign($r_i$, $r_j$)</td>
<td>$r_i, r_j \in \text{Scope}_R(\text{ar})$</td>
<td>$(r_i, r_j) \in \text{RH}$</td>
</tr>
<tr>
<td>Revoke($r_i$, $r_j$)</td>
<td>$r_i, r_j \in \text{Scope}_R(\text{ar})$</td>
<td>$(r_i, r_j) \notin \text{RH}$</td>
</tr>
<tr>
<td>Assign(u, g)</td>
<td>$u \in \text{Scope}_U(\text{ar}), g \in \text{Scope}_G(\text{ar})$</td>
<td>$(u, g) \in \text{UG}$</td>
</tr>
<tr>
<td>Revoke(u, g)</td>
<td>$u \in \text{Scope}_U(\text{ar}), g \in \text{Scope}_G(\text{ar})$</td>
<td>$(u, g) \notin \text{UG}$</td>
</tr>
<tr>
<td>Assign(u, r)</td>
<td>$u \in \text{Scope}_U(\text{ar}), r \in \text{Scope}_R(\text{ar})$</td>
<td>$(u, r) \in \text{UR}$</td>
</tr>
<tr>
<td>Revoke(u, r)</td>
<td>$u \in \text{Scope}_U(\text{ar}), r \in \text{Scope}_R(\text{ar})$</td>
<td>$(u, r) \notin \text{UR}$</td>
</tr>
<tr>
<td>Assign(p, r)</td>
<td>$p \in \text{Scope}_P(\text{ar}), r \in \text{Scope}_R(\text{ar})$</td>
<td>$(p, r) \in \text{PR}$</td>
</tr>
<tr>
<td>Revoke(p, r)</td>
<td>$p \in \text{Scope}_P(\text{ar}), r \in \text{Scope}_R(\text{ar})$</td>
<td>$(p, r) \notin \text{PR}$</td>
</tr>
<tr>
<td>Assign(p, g)</td>
<td>$p \in \text{Scope}_P(\text{ar}), g \in \text{Scope}_G(\text{ar})$</td>
<td>$(p, g) \in \text{PG}$</td>
</tr>
<tr>
<td>Revoke(p, g)</td>
<td>$p \in \text{Scope}_P(\text{ar}), g \in \text{Scope}_G(\text{ar})$</td>
<td>$(p, g) \notin \text{PG}$</td>
</tr>
</tbody>
</table>
ACaaS\textsubscript{RBAC} Design

- **Architecture**
  - ACaaS\textsubscript{RBAC} introduces RBAC as a service (RaaS), which is an RBAC module can be hosted by AWS or any third party service provider.
ACaaS<sub>RBAC</sub> Design – Cont'd

- RaaS consist of 8 sub-modules, each of which is exposed as a web service
  - *Organization module*: manage (e.g., list, register, and delete) organizations to support the multi-tenant feature
  - *Group module*: manage user groups $G$ of a single organization for administrative users $AU$
  - *Permission module*: manage permissions $P$, $AP$, and their relations with roles $R$, $AR$
  - *User module*: manage (e.g., create, delete, activate, deactivate) users $U$, $AU$, and their relations with roles $R$, $AR$

  Note: both user and permission info are directly provisioned by AWS for existing AWS customers.
ACaaS\textsubscript{RBAC} Design – Cont'd

- **Role module**: manage (e.g., create, delete) roles $R$, $AR$, and $RH$
  - Towards efficient role-related operations, we adopt the Nested Set Model
  - It assigns left and right values to represent a scope of each role in a role hierarchy

- **Session module**:
  - Support efficient role activation and deactivation operations
ACaaS\textsubscript{RBAC} Design – Cont'd

○ role activation and deactivation algorithms

\textbf{Algorithm 1:} ComputeActivatePermissions\((u, r_a) \rightarrow P\)

\begin{itemize}
\item \textbf{Input:} A user \(u\) wants to activate a role \(r_a\)
\item \textbf{Output:} A permission set \(P\), of which corresponding IAM policies need to be enforced
\end{itemize}

\begin{algorithmic}
\STATE \(P \leftarrow \emptyset;\)
\STATE \(P_{all} \leftarrow \emptyset;\)
\STATE \(r_{ts} \leftarrow \text{getImmediateSeniorRole}(r_a);\)
\IF {hasRole\((u, r_{ts}) = \text{TRUE} \AND \text{active}(u, r_{ts}) = \text{TRUE}\)}
\STATE \text{return } \emptyset;\)
\ELSE
\STATE \text{ComputeP}\((u, r_a);\)
\FORALL {\(p \in \text{Permissions}(r_a)\)}
\IF {\(p \notin P_{all}\)}
\STATE \text{add } p \text{ into } P_{all} ;\)
\ENDIF
\ENDFOR
\ENDIF
\STATE \text{return } P_{all};\)
\STATE \text{ComputeP}\((\text{User } u, \text{ Role } r_a)\)
\STATE \text{begin}
\STATE \(R \leftarrow \text{getSiblingRoles}(r_a) \cup \text{getImmediateJuniorRoles}(r_a);\)
\IF {\(R = \emptyset\)}
\STATE \text{return } \emptyset;\)
\ELSE
\FORALL {\(r \in R\)}
\IF {\text{active}(u, r) = \text{TRUE}}
\FORALL {\(p \in \text{Permissions}(r)\)}
\IF {\(p \notin P_{all}\)}
\STATE \text{add } p \text{ into } P_{all} ;\)
\ENDIF
\ENDFOR
\ENDIF
\STATE \text{else computeP}\((u, r);\)
\STATE \text{end}
\STATE \text{end}
\STATE \text{return } P;\)
\STATE \text{end}
\end{algorithmic}

\textbf{Algorithm 2:} ComputeDeactivatePermissions\((u, r_d) \rightarrow P\)

\begin{itemize}
\item \textbf{Input:} A user \(u\) wants to deactivate a role \(r_d\)
\item \textbf{Output:} A permission set \(P\), of which corresponding IAM policies need to be enforced
\end{itemize}

\begin{algorithmic}
\STATE \(P \leftarrow \emptyset;\)
\STATE \(P_{all} \leftarrow \emptyset;\)
\STATE \(R_{senior} \leftarrow \text{getSeniorRoles}(r_d);\)
\IF {\(R_{senior} \neq \emptyset\)}
\FORALL {\(r \in R_{senior}\)}
\IF {\text{active}(u, r) = \text{TRUE}}
\STATE \text{return } \emptyset;\)
\ENDIF
\ENDFOR
\ENDIF
\STATE \text{R sibling} \leftarrow \text{getActivatedSiblingRoles}(r_d);\)
\IF {\text{R sibling} = \emptyset;\)
\STATE \text{return } \text{Permissions}(r_d);\)
\ELSE
\FORALL {\(r \in \text{R sibling}\)}
\IF {\text{active}(u, r) = \text{TRUE}}
\FORALL {\(p \in \text{Permissions}(r)\)}
\IF {\(p \notin P_{all}\)}
\STATE \text{add } p \text{ into } P_{all} ;\)
\ENDIF
\ENDFOR
\ENDIF
\STATE \text{else computeP}\((u, r);\)
\STATE \text{end}
\STATE \text{end}
\STATE \text{return } P;\)
\STATE \text{end}
\end{algorithmic}
ACaaS\textsubscript{RBAC} Design – Cont'd

- role activation and deactivation algorithms example
ACaaS$_{RBAC}$ Design – Cont'd

- **Constraint module**: provides constraints management services including creating, deleting, updating static constraints as well as SoD
  - Static constrains are specified based on IAM syntax, and enforced by IAM
  - SoD is enforced by constraint module

- **Policy module**: provides Amazon IAM policy generation and pushing services to ensure RBAC configurations of an enterprise can be reflected in AWS cloud platform
  - For example, for each permission in the permission set computed by Algorithm 1 or Algorithm 2, a corresponding IAM policy is constructed and sent to IAM for the enforcement
Implementation

- Implemented a prototype system to provide RBAC services in AWS cloud platform through a web browser interface as well as web services.
- The core services of the system are implemented in Java based on AWS SDK 1.3.0 and exposed as web services using GlassFish Metro 2.2.
- The web-based management interface is developed by using JavaServer Pages (JSP).
Evaluation

- We evaluated our ACaaS\textsubscript{RBAC} system in terms practicality, efficiency and scalability
  - A scenario-based case study is presented to illustrate how services provided by ACaaS\textsubscript{RBAC} can be utilized by cloud applications
  - We then show the performance measured on role activation and deactivation services
Evaluation – Cont'd

- A Sandbox scenario for software development in AWS with ACaaS\textsubscript{RBAC}

```json
{ "Statement": [{
   "Sid": "Stmt1360134990238",
   "Action": [
      "ec2:CreateKeyPair",
      "ec2:DescribeInstances",
      "ec2:ImportKeyPair"],
   "Effect": "Allow",
   "Resource": "*"}
}

Policy_2

{ "Statement": [{
   "Sid": "Stmt1360133406863",
   "Action": [
      "s3:GetObject",
      "s3:GetObjectVersion",
      "s3:PutObject"],
   "Effect": "Allow",
   "Resource": "arn:aws:s3:::S3-B1"
}
}

Policy_1

Policies

---

The Laboratory of Security Engineering for Future Computing

AWS Resources

DSE1

IAM

RDS-SI1

EC2-CI1

EC2-CI2

----

DSE2

...
Evaluation – Cont'd

- Alice is a member of the role Dev1 working on Project 1 and Bob is a member of the role Dev2 working on Project 2.
- In order to complete collaborative task, Alice need to be assigned to role Dev2.
- We measure the Dev2 role activation for Bob when he logs in and the Dev2 role activation for Alice when she is assigned to the role Dev2 in terms of performance overhead (T) and network traffic overhead (NT).

Results:

<table>
<thead>
<tr>
<th></th>
<th>T (ms)</th>
<th>PG (ms)</th>
<th>PTE (ms)</th>
<th>NT (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev2 Activation for Bob</td>
<td>1898.9</td>
<td>91.6</td>
<td>1807.2</td>
<td>15.95</td>
</tr>
<tr>
<td>Dev2 Activation for Alice</td>
<td>1200.2</td>
<td>47.4</td>
<td>1152.4</td>
<td>14.652</td>
</tr>
</tbody>
</table>
In order to measure scalability of ACaaS\textsubscript{RBAC}, measure average performance overhead while increasing the numbers of simultaneous role activation and deactivation requests from users.
Related Work

Authorization and access control solutions in clouds

- An authorization model that supports multi-tenancy, role-based access control, path-based object hierarchies, and federation [Calero’10]
- Semantic web technologies to distributed role-based access control method and propose a new Semantic Access Control Policy Language (SACPL) for describing access control policies in clouds [Hu’09]
- CloudPolice, a distributed access control mechanism implemented in hypervisors, to meet the access control needs of multi-tenancy, network-independence, and scalability in cloud computing environment [Zhu’12]

Cryptographic techniques based access control in clouds

- An effective solution for de-coupling access control from services that provide content by leveraging ABE [Echeverria’10]
- Design of a trusted cloud computing platform (TCCP) [Santos’09]
- Use proxy reencryption and broadcast revocation algorithms for flexible content access control in cloud [Zhang’12]
- HASBE scheme which incorporates a hierarchical structure of system users by applying a delegation algorithm to ASBE [Wan’11]
Conclusion

- We articulated the critical need of a comprehensive and fine-grained access control mechanism to meet dynamic, configurable, and extensible security requirements in public IaaS cloud computing environments.
- To accommodate this need, we proposed a new modular architecture towards access control as a service (ACaaS) for supporting multiple access control models.
- As a reference implementation, we designed and implemented ACaaSRBAC, a service architecture that supports configurations of RBAC as a service for AWS.
- Our case study and system performance evaluation demonstrated the practicality and efficiency of our approach.
Future Work

- We would evaluate our system with real world datasets. In addition, we would enhance our system with flexible delegation mechanisms and accommodate revocation requirements, and design a more generic architecture to support other access control policies such as multilevel and general mandatory access control policies.
Thank you !!!