2021 International Conference on Big Data Engineering and Education (BDEE) | 978-1-6654-3957-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BDEE52938.2021.00038

HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager

Chittipat Pasomsup
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
cpasomsup@outlook.com

Yachai Limpiyakorn
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
Yachai.L@chula.ac.th

***Abstract*—For transitioning to a decentralized system, a microservices platform has become popular in today software development due to its lightweight mechanisms. However, increasing the number of services results in a challenge to maintain the security of access control. The more attack surfaces can bring security and privacy risk via sensitive data. Therefore, a chain of trust domains was introduced to solve this problem. The extended Role-Based Access Control model (Hierarchical Trust RBAC: HT-RBAC) for microservice security managers is proposed for leveraging threats of unauthorized access to sensitive information and identity verification across all environments in application container solutions. This paper proposes using an HT-RBAC to authenticate, authorize, and identify user's access control via API-Gateway. A prototype system integrated with OAuth 2.0 authentication server is implemented for empirical study. The results report that the approach provides faster and more flexible access to information in addition to improving incident response time.**

Keywords—role-based access control, IT security, identity management, microservices

I. Introduction

Nowadays, the development of applications within the organization is growing rapidly. Microservices architecture has been introduced as an alternative solution for scalability and elasticity of the system infrastructure as opposed to the old monolithic architecture. The approach defines the process of segregating the services available in the monolith into a set of independent services, or a modular structure. In comparison with Monolithic design, a change to an application requires the whole system to be rebuilt and deployed. For this reason, maintainability is harder and inefficient. The microservice is Service Oriented Architecture (SOA) which helps a rapidly connection and lightweight delivery. And mostly practical implication of microservices is in the cloud computing industry [1]. Due to the large number of services to operate with, access control is one of the difficult problems to manage, especially for large organizations which often have complex data storage systems. The controlling access to resources is crucial and challenging to prevent the disclosure of stored data or confidential business information. Therefore, different types of access control have been studied, one of which is the well-known Role Based Access Control (RBAC), which determines the rights of users based on their responsibilities. The prioritization of officers with the access to various types of sensitive information, such as transactional, financial, or personal information of trading partners, etc., is essential. In order to prevent bad people from taking this information publicly, establishing a RBAC Model [2] for identity management and access control is one of the issues that organizations pay attention to. Additionally, the security challenges of microservices are the increasing attack surface of applications and unknow vulnerability. This paper presents the formal RBAC adapted from a centralized model to decentralized model that suits for multiple trusted domains. Our hierarchical role-based access control model could be applied between each microservice due to inheritance relation which is less complex than the attribute-based model. This work contributes a design of dynamic RBAC that supports a stateless authentication often used in microservice applications. The rest of paper is organized as follows. Section 2 describes the background knowledge relating to the RBAC model and Microservice architecture. Section 3 details the proposed method. The case study is demonstrated in Section 4. Finally, the conclusion and further direction are stated in Section 5.

II. Background

A. *The Core and Hierarchichical Role Based Access Control Model*

The RBAC reference model and the RBAC System and Administrative Functional Specification is a standard created by American National Standard for Information Technology in 2004 [2]. This model could solve an access revocation complexity problem and easy to classify users in practical implementation. The more complexity which may not depend on a role based model could lead to add more attributes that would cause an inaccurate rule. This requires an expert to fix that exhaustive time in DevOps. The core model of RBAC consists of five data elements, which are users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). The relationship can be assigned as an individual or a group of elements, that are many to many relationships as illustrated in Fig. 1. The user assignment (UA) and permission assignment (PA) relations show semantic constructs that provide granularity of assignment of permission form roles and users to roles. For example, users in the same department whose are not own file, can only view that file with a share permission records. This access control shows flexibility and the principle of least privilege.

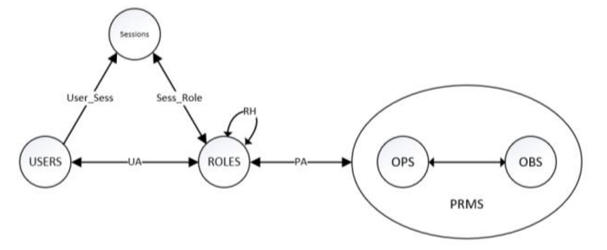


Fig. 1. The Core and Hierarchichical RBAC model [2].

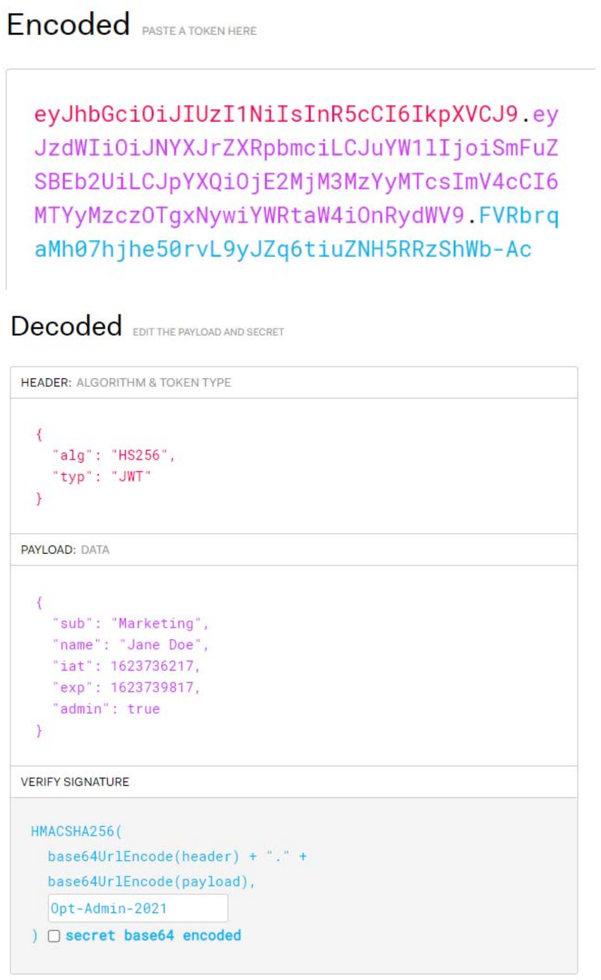
The basic specifications of this model are defined:

- *USERS, ROLES, PRMS,* and *Sessions* (users, roles, permissions and sessions respectively)
- *PA* $\subseteq$ *PRMS* × *ROLES*, is a many-to-many permission to role assignment relation.
- *UA* $\subseteq$ *USERS* × *ROLES*, is a many-to-many user to role assignment relation;
- RH $\subseteq$ *ROLES* × *ROLES*, is a partial order on *ROLES* called the role hierarchy or role dominance relation, written as $\precsim$

In 2013, Zhu et al. [3] presented cryptographic hierarchical RBAC model, an elliptic curve cryptosystem was introduced to support signature and authentication. In 2015, Zhu et al. [4] extended RBAC to ABAC via attributed-based encryption. The attribute hierarchies help a light weight ciphertexts and private keys. The work of Fugkeaw et al. [5] focused on dynamic RBAC models. The authors present a design of development for Single Sign-On (SSO) with two-factor authentication in dynamic RBAC models, that work on multi-applications and multi-domain systems. In [6], the authors present a dynamically capable RBAC+ model which helps improving security of databases by detecting and terminating malicious transactions.

B. *Identification, Authentication, and Authorization in Microservices*

A scalable Microservice is a combination of agile development that would deliver applications and services at evolving and improving products at a faster pace [7]. It requires an application container (Docker) to gather all service works together, whereas REST architectural style is a common communication for microservices. In term of security concept in microservices, the core function is identification. Due to its decentralized system, a delegated system, Security Manager, is required to operate among each service. The other important functions of this management system include authentication and authorization that use Authorization Server (OAuth 2.0) [8] to access microservices via API-Gateway. The access token is encrypted in Jason Web Token (JWT) passing to validate the user identity, and granting the user session, respectively [9]. JWT is an open standard (RFC 7519) which uses securely transmitting data between services. Its structure consists of Header, Payload and Signature. The cryptographic algorithm normally used is HMAC SHA256 and encoded to Base64Url form as shown in Fig. 2. The challenge of implementation is strengthened security because of various number of services. An alternative solution is a unified authentication and authorization as presented in [10]. In other works [11,12], the authors proposed the approaches based on distributed system with collaboration between microservices. Instead, we propose a model to specify an integrated system of their approach that can unify identity management which is composed of security microservices cloud to operate in multi-environment.


Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJNYXJrZXRpbmciLCJuYW1lIjoiSmFuZ
SBEb2UiLCJpYXQiOjE2MjM3MzYyMTcsImV4cCI6
MTYyMzczOTgxNywiYWRtaW4iOnRydWV9.FVRbrq
aMh07hjhe50rvL9yJZq6tiuZNH5RRzShWb-Ac
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "Marketing",
  "name": "Jane Doe",
  "iat": 1623736217,
  "exp": 1623739817,
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Opt-Admin-2021
) ☐ secret base64 encoded
```

Fig. 2. Use of JWT for authorization (jwt.io debugger).

III. Hierarchical Trust RBAC Model (HT-RBAC)

Referring to ANSI INCITS [2], the standard describes the components and relationships of the RBAC model, including the role hierarchy. And limitations so as not to cause the model to conflict with each other are assigning roles when the environment of use. There is a greater complexity and diversity of resources to manage, the models are thus classified into 3 types: core, hierarchy, and constraint. Core RBAC consists of the sets USERS, ROLES, SESSIONS and PRMS that represent the sets of users, roles, sessions, and permissions, respectively. The permissions set consists of two domains which are Objects (OBS) and Operations (OPS). The two-part relationship, User Assignment (UA), is the relationship between the user to the role, and the Permission Assignment (PA) is the relationship between the roles to the permissions for various objects. The design of HT-RBAC model is depicted in Fig. 3.

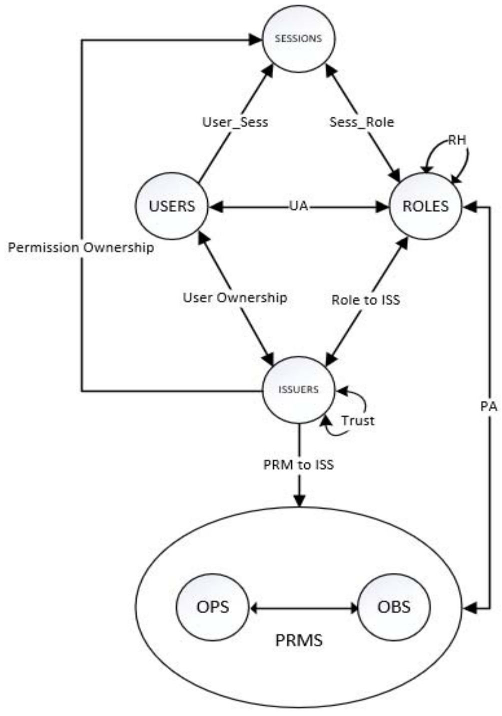


Fig. 3. Design of HT-RBAC model for Microservice Security Manager.

The Security Manager (SM) is used for accounting and identity management. It is an application used for managing authentication sessions which grant authorization to user roles. The SM's Issuers are conducted to control multi-services of the authorization system. The relationship added to this model is an Issuer Trust ($IT \subseteq I \times I, \forall in$). The client using these services will create an interface for each issuer, so the individual information and actions of each issuer will be explicitly separated from the other issuer. In addition, the IT relationship has a Role Hierarchy (RH) which represents a chain of trust of each service correlated to that user. And Permissions (PRMS) is authorized by the issuer, to gain access to their object which will work through an interface. The authorization consists of three parts: rights, operation, grant objects. The granting of one user is subject to only one issuer, and every ownership relates to only one issuer, but more than one issuer can also be granted. A role is a function that is the name of the duty of the issuer, where the role representation consists of role (issuer, role name) where one role can be defined for only one issuer, but one issuer can have many roles such as together. Then session is a constant value in which the various activities of user access are stored and restricted by permission ownership.

The relationships of each element are defined as following:

- *Role to ISS* $\subseteq$ *ROLES* $\times$ *ISSUERS*, a many-to-many mapping Role-to-Issuers assignment relation.
- *ISSUERS_roles* : *ISSUERS* $\rightarrow$ *ROLES*, the mapping of an issuers to a set of roles.
- *ISSUERS_roles (iss)* = {*r* $\in$ *ROLES*|(*r, iss*) $\in$ *Role to ISS* }.
- *Permission Ownership : SESSIONS* $\rightarrow 2^{Permission\ Ownership}$. The mapping of an issuers onto a set of sessions
- *PRMS* = $2^{(OPS \times OBJ)}$
- *PRM to ISS* $\subseteq$ *PRMS* $\times$ *ISSUERES*, a many-to-many mapping Permission-to- Issuers assignment relation.
- *ISSUERS_perms* : *ISSUERES* $\rightarrow 2^{PRMS}$, the mapping of an Issuers onto a set of permissions.
- *ISSUERS_perms* (iss) = {*p* $\in$ *PRMS*|(*p, iss*) $\in$ *PRM to ISS*}
- *avail_session_perms(s : SESSIONS)* $\rightarrow 2^{PRMS}$, the permissions available to a user in a session = $\cup_{r \in \text{session roles}(s)}$ assigned permissions(r)
- *session(s$_i$)* = {*as$_{ij}$* |*j* = *1, 2, ...n*} as stands for Issuer Session. A session is composed of many issuers sessions.
- *User Ownership* $\subseteq$ *ISSUERS* $\times$ *USERS*, a many-to-many mapping application-to-user assignment relation.
- *USER_AssignedIssuers : USERS* $\rightarrow 2^{ISSUERS}$, the mapping of a user to a set of Issuers.
- *USER_AssignedIssuers (u)* = {*u* $\in$ *USERS*|(*iss, u*) $\in$ *User Ownership* }.
- *session_user: SESSIONS* $\rightarrow$ *USERS*, the mapping from a session s to the user of s.
- *session_roles : SESSIONS* $\rightarrow 2^{ROLES}$, the mapping of session *s* onto a set of roles.
- *session_roles(s)* $\subseteq$ {*r* $\in$ *ROLES*|(*session_user(s), r*) $\in$ *UA*}.
- *avail_iss_roles* : (*SESSIONS, ISSUERS*) $\rightarrow 2^{ROLES}$, the mapping of a session and an issuer onto a set of roles.
- *avail_iss_roles (s, iss)* $\subseteq$ {*r* $\in$ *ROLES*| *r* = *session_roles(s)* $\cap$ *ISSUERS_roles (iss)* }
- *avai_iss_prms : (SESSIONS, ISSUERS)* $\rightarrow 2^{PRMS}$, the permissions available to an issuer in a session.

  avai_iss_prms (*s, iss*) = $\cup_{r \in avail\ app\ roles(s,app)}$ *assigned permissions(r)*

IV. Case Study

The design of RBAC for Microservice security manager is implemented. The prototype demonstrates the use of HT-RBAC, assuming that the marketing organization is in Software as a Service (SaaS) and has four microservices: promotion, selling, product management, and distribution. The working environment is established using Node.JS, MongoDB, and Express framework. The microservice communicates others via HTTP protocol and JSON Web Token (JWT) over SSL/TLS which encrypts data for more security. REST API is adopted for inter-services and processes. The implementation of API-Gateway style uses a centralization for all the clients. The combination of microservices architecture and API-Management provides better cooperation with microservice security manager to facilitate identity management. Fig. 4 illustrates an overview of the deployment of the microservices application. The express-session-jwt module is imported to help token-based authentication. It uses JWT as session tokens that can store session ID with the user and role. This middleware can manage API and serial generate sessions via trust proxy. Fig. 5 shows the diagram of basic process flow of JWT-based authentication. Each microservice instance is deployed as a container. To secure each microservice with an access delegation protocol (OAuth 2.0), the client authenticates with the authorization server and gets an access token then issues a user to grant a session.

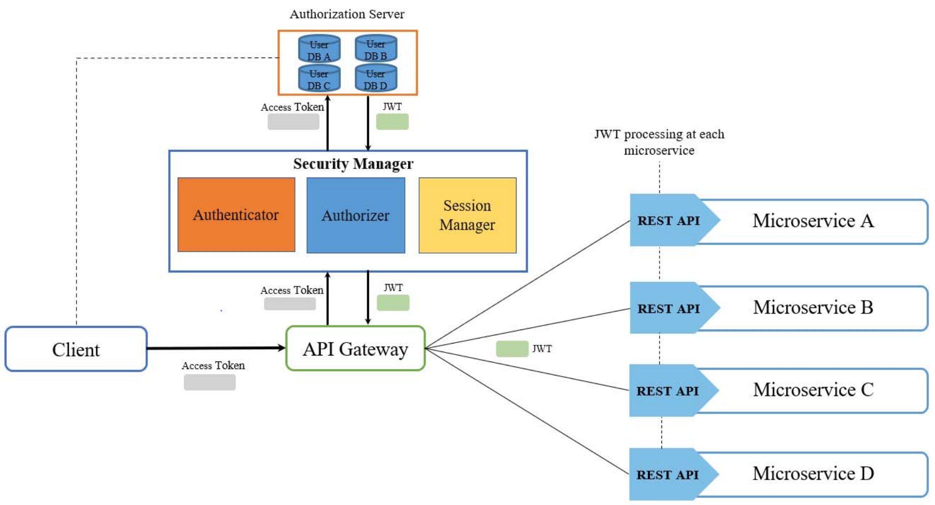


Fig. 4. Test scenario for prototyping microservice security manager.

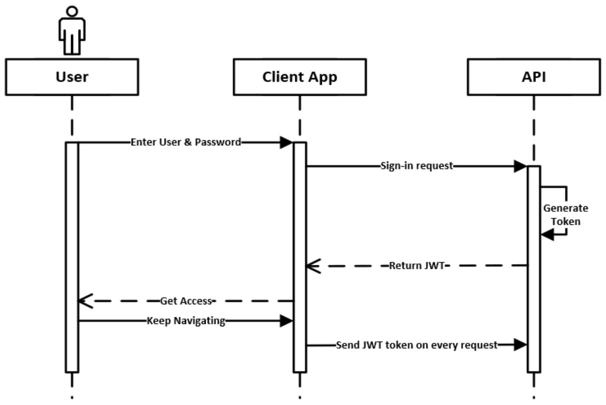


Fig. 5. Client Credentials Flow in JWT authentication.

The Security Manager contains three services consisting of Authenticator, Authorizer, and Session Manager. Based on HT-RBAC model, the Authenticator gets USERS and access token then sends it to verify with OAuth sever. The Authorizer receives a reply JWT in Base64 encoding and RS256 encryption of USERS, ROLES, OPS, and OBJ then executing permission to Session Manager to give a session corresponding to that user.

Based on the presented RBAC model, the whole operation can be completed to prevent an exploitation due to the lack of authentication and authorization mechanisms. The Microservice Security Manager is created to manage Create, Read, Update, and Delete (CRUD) functionality of users, and roles. It describes a remove permission function that would occur in request for change process. A prototype system is written in TypeScript. Example of source code is shown in Fig. 6 which describes a private method to remove an action of permission assignment on a specific role. It could check the object and role that is not empty and validate an operation owned by role then remove an object associated with that operation.

```
RBAC.prototype._removePermission = function (objects,
roles, actionOwnership) {
    var _this = this;
    objects = utils_1.utils.toStringArray(objects);
    if (objects.length === 0 || !utils_1.utils.
    isFilledStringArray(objects)) {
        throw new core_1.AccessControlError(
        "Invalid object(s): " + JSON.stringify(objects));
    }
    if (roles !== undefined) {
        roles = utils_1.utils.toStringArray(roles);
        if (roles.length === 0
        || !utils_1.utils.isFilledStringArray(roles)) {
            throw new core_1.AccessControlError
            ("Invalid role(s): " + JSON.stringify
            (roles));
        }
    }
    utils_1.utils.eachRoleObject(this._grants, function
    (role, object, permissions) {
        if (objects.indexOf(object) >= 0
            && (!roles || roles.indexOf(role) >= 0)) {
            if (actionOwnership) {
                var ao =
                utils_1.utils.normalizeActionOwnership(
                {action: actionOwnership}, true);
                delete _this._grants[role][object][ao];
            }
            else {
                delete _this._grants[role][object];
            }
        }
      });
};
```

Fig. 6. Remove Permission function in rbac.js.

V. Conclusion and Future work

This paper presents a design of HT-RBAC model, which introduces a trust hierarchy able to operate across multi-domain of microservices. Integrating the Security Manager to the model could help authenticate, authorize, and identify user's access control. A prototype system is implemented to demonstrate the proposed approach. The testing API gateway scenario reported that the model can operate properly. Moreover, the chain of trust in each service are bound with the user to ensure that none of cross-domain requests could lead to Cross-Site Request Forgery (CSRF) or Cross-Site Scripting (XSS) attack. Further evaluation such as verification of a test case, and the analysis of model performance would be carried out. The gRPC should be conducted to compare with REST APIs. To strengthen higher security, a two-factor authentication or advance encryption algorithms is suggested to ensure reliability and confidentiality.

References

[1] N. Dragoni, S Giallorrnzo, A. Lafuente, et al., "Microservices: Yesterday today and tomorrow". Present and Ulterior Software Engineering, no. 4, pp. 195-216, 2017.

[2] American National Standard for Information Technology Role based Access Control. ANSI incits 359-2004

[3] Y. Zhu, G. Ahn, H. Hu, D. Ma and S. Wang, "Role-Based Cryptosystem: A New Cryptographic RBAC System Based on Role-Key Hierarchy," in IEEE Transactions on Information Forensics and Security, vol. 8, no. 12, pp. 2138-2153, Dec. 2013.

[4] Y. Zhu, D. Huang, C. Hu and X. Wang, "From RBAC to ABAC: Constructing Flexible Data Access Control for Cloud Storage Services," in IEEE Transactions on Services Computing, vol. 8, no. 4, pp. 601-616, 1 July-Aug. 2015.

[5] S. Fugkeaw, P. Manpanpanich and S.Juntapremjitt, "A development of multi-SSO authentication and RBAC model in the distributed systems," *2007 2nd International Conference on Digital Information Management*, Lyon, France, 2007, pp. 297-302

[6] A. Bouchahda, N. L. Thanh, A. Bouhoula and F. Labbene, "RBAC+: Dynamic Access Control for RBAC-Administered Web-Based Databases," *2010 Fourth International Conference on Emerging Security Information, Systems and Technologies*, Venice, Italy, 2010, pp. 135-140

[7] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. **33**, no. 3, pp. 42-52, May-June 2016

[8] OpenID AB/Connect, OpenID Connect 1.0 specification. https://openid.net/connect/ (2014) Accessed Mar 2021

[9] K. Indrasiri, "Microservices in Practice," http://kasunpanorama.blogspot.com/2015/11/microservices-in-practice.html (2015). Accessed Mar 2021

[10] Y. ShuLin and H JiePing, "Research on Unified Authentication and Authorization in Microservice Architecture," 2020 *IEEE 20th International Conference on Communication Technology (ICCT)*, Nanning, China, 2020, pp. 1169-1173

[11] B. Tang, R. Sandhu and Q. Li, "Multi-tenancy authorization models for collaborative cloud services," *2013 International Conference on Collaboration Technologies and Systems (CTS)*, San Diego, CA, USA, 2013, pp. 132-138

[12] D. Thongsang and Y. Temtanapat, "A chain calling in coordination for multi-tenant collaborative cloud services," *2014 International Computer Science and Engineering Conference (ICSEC)*, Khon Kaen, Thailand, 2014, pp. 302-307