

Hybrid Role and Attribute Based Access Control Applied in Information Systems

Maria Penelova

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria

E-mail: i_n_f@abv.bg

Abstract: *In this paper it is proposed a new access control model – Hybrid Role and Attribute Based Access Control (HRABAC). It is an extension of Role-Based Access Control (RBAC). HRABAC is designed for information systems and enterprise software and combines the advantages of RBAC and Attribute-Based Access Control (ABAC). HRABAC is easy configurable, fine-grained and supports role hierarchies. The proposed model HRABAC describes the access control scheme in Laravel package laravelroles/rolespermissions, which is developed by the author of the paper, as an answer to the requirements of practice of fine-grained and easy configurable access control solution. Laravel is chosen, because it is the most popular and the most widely used PHP framework. The package laravelroles/rolespermissions is developed on Laravel so that maximum number of programmers could use it. This package contains working and tested functionalities for managing users, roles and permissions, and it is applied in accounting information system.*

Keywords: *Access control, authorization, access decision, HRABAC, hybrid role and attribute access control.*

1. Introduction

Information technology security consists of authentication [8, 9] authorization [9], and audit [7, 9]. Authentication is the way of proving the identity of a user, usually via entering the credentials of the user. Authorization determines whether to grant or deny access to a resource. Another term for authorization is Access Control (AC). The formal representation of authorization is AC model. An AC model consists of set of subjects, and set of objects and policies. A subject is mapping of user, which is assigned to many roles or attributes. The subject is an active entity. An object is the requested resource and it is a passive entity. The policy is a rule, which decides whether to grant or deny the access of a subject to object.

Granularity is a characteristic that refers to the ability of an AC model to provide a detailed check for access control purposes. If an AC model does not have an ability to provide a detailed check, then it is coarse-grained AC model. If the AC model does additional checks, like a verification whether a user is the owner of the requested object, then the model is fine-grained.

RBAC [2] and ABAC [5] are AC models, widely used in enterprise software. RBAC uses roles in its policies.

RBAC has easy for maintaining policy and it is easy to be configured. Limitation of RBAC is that it is coarse-grained.

Using attributes provides to ABAC better granularity than RBAC. Policies with attributes check in more detail, that is why ABAC is fine-grained. But the complex policies and tree-based structure of attributes are hard to be maintained and configured, and have become disadvantages of ABAC.

There is a trend to combine RBAC and ABAC, in order to use the advantages and to avoid the drawbacks of these two AC models for enterprise software and to provide fine-grained and easy configurable AC model [1].

Nowadays, AC models are created in areas, such as blockchain [12, 14, 15, 17, 20], Internet of things [14], cloud computing [16, 22, 23] and distributed systems [11]. Fine-grained AC models [13, 19] are introduced. Lately, there are researched dynamic AC schemes, based on RBAC [10, 12] and models, which are based on ABAC [14, 21], as well as hybrid models [18] for big data.

Personal data should not be shown to all people indiscriminately. Such data in information systems should be protected via fine-grained control. The requirements of the practice for an easily configurable and fine-grained access control model for information systems and enterprise software, are still not satisfied.

This motivates the author of this paper to create a new model for access control. Thus, Hybrid Role and Attribute Based Access Control (HRABAC) is proposed. It is designed for information systems and enterprise software and is an extension of RBAC. HRABAC preserves the easy configurability of RBAC and the expressive power of ABAC policies. The proposed model describes the access control scheme that is implemented in Laravel package `laravelroles/rolespermissions` [25, 26]. This package is developed by the author of the paper.

An advantage of HRABAC is the easy to maintain policy. It prevents an unauthorized action to be executed and, at the same time, protects certain data. In information system, the authorization process in HRABAC regulates the execution of actions, like view and edit, against a database record. Meanwhile, it protects a database record, when an action is granted, via additional checks. An application of the proposed model is given in an accounting information system.

The rest of this paper is structured as follows. In Section 2, the AC models for enterprise software – RBAC and ABAC, are described. In Section 3, the new model HRABAC is proposed. The application of `laravelroles/rolespermissions`, that is based on HRABAC is described in Section 4. In Section 5 a comparison between ABAC, RBAC and HRABAC is proposed. The results are presented in a table. Section 6 suggests future developments and conclusions.

2. Related work

RBAC [3, 4, 24] is the most popular AC model for enterprise software. The family of RBAC models – RBAC96 framework is introduced in 1996 [4]. It consists of four models: the basic RBAC₀ model, the advanced models RBAC₁ and RBAC₂, and the consolidated model RBAC₃.

The basic RBAC₀ model has three sets of entities as components: a set of users, a set of roles and a set of permissions. A user is human being. A role is a job title within organization. A permission is approved access to resources. They are always positive. A session is a mapping of one user onto a subset of roles. The subsets of used roles in different sessions may differ. That supports the least privilege principle. Permissions that modify the sets of users, roles and permissions are called administrative permissions.

The model RBAC₁ supports role hierarchies. This is the ability of one role to inherit the permissions of another role.

RBAC₂ introduces constraints. RBAC₃ combines RBAC₁ and RBAC₂.

NIST (National Institute of Standards and Technology) ABAC [5, 24] has been introduced in 2014. The components of ABAC [6] are subjects, objects, the attributes of subjects, the attributes of objects, environment conditions, policies and AC mechanism. The AC mechanism consists of policy decision point and policy enforcement point. A subject is human being, computer or router. An object is the requested system resource. A policy is a rule that regulates the access of the user, using the values of subject attributes, object attributes and environment conditions. The AC mechanism evaluates the subject attributes, the object attributes, the environment conditions and policies, and determines whether to grant or deny access.

Time and location can be involved in ABAC policy as environment conditions. ABAC policy can be expressed for attributes that not exist at the moment, but they can be added in future.

There are many software solutions, based on RBAC, called installable packages for managing roles and permissions [27-29]. They possess working and tested AC schemes that are installed on a developed application, in order to save time and effort of the software developers. The packages have the limitations of the AC model RBAC, so they are not fine-grained.

ABAC is too complicated for configuring to be implemented in an installable package. There are no packages, based on ABAC.

In response to the requirements of the practice for an easily configurable and fine-grained access control solution that can be installed on a developed information system, Laravel package `laravelroles/rolespermissions` [25, 26] has been developed. Laravel is the most popular PHP framework and the most widely used. The package `laravelroles/rolespermissions` has been developed on Laravel, so that the maximum number of programmers could use it. It provides working and tested functionalities for managing users, roles and permissions, which are added to an information system only via installation, there is no necessity for writing programming code. HRABAC has the advantages of both RBAC and ABAC models that are easy to configure and possess fine-grained access control.

3. HRABAC

It is important to combine RBAC and ABAC, in order to use the advantages and to avoid the drawbacks of both AC models for enterprise software [1]. Assume we have an accounting information system with four roles: Administrator, Manager, Accountant and Employee. Administrator has access to all pages of the information system. Manager can manage (create, edit, delete, read) employees and the salaries of employees. Accountant can manage salaries of employees. Employee can view only his/her salary data. The roles: Administrator, Manager, Accountant and Employee, can be assigned to users with RBAC.

The problem in RBAC is: if to the role Employee is given permission to read salary data, it would be possible the user with role Employee to see the salary data of all users in the database. If the user with the role Employee is forbidden to see the salaries of all users, he/she cannot see his/her own salary data. But the requirement of the system is: the users with role Employee to see only his/her salary data.

Here can be used the expressive power of ABAC policy. For that purpose, we introduce policy functions that evaluate attributes. Thus, HRABAC model is designed to cover the security requirements of information systems.

Definition 1. Let $Users = \{u_1, u_2, \dots, u_n\}$ be a set of users, $Roles = \{r_1, r_2, \dots, r_m\}$ be a set of roles, Ops be a set of operations, Obs be a set of objects, $Permissions = \{p_1, p_2, \dots, p_q\} = 2^{(Ops \times Obs)}$ is a set of permissions.

Definition 2. Let $Subjects$ be a set of subjects, where subject is a mapping of one user to many roles.

Definition 3. The function $assigned_permissions: Roles \rightarrow 2^{Permissions}$ is the mapping of role r onto a set of permissions.

Definition 4. The function $subject_roles: SUBJECTS \rightarrow 2^{Roles}$ is the mapping of subject s onto a set of roles.

Definition 5. $RH \subseteq Roles \times Roles$ is a partial order on $Roles$ called inheritance relation, written as \geq , where $r_1 \geq r_2$ only if all permissions of r_2 are also permissions of r_1 , and all users of r_1 are also users of r_2 .

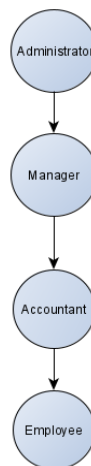


Fig. 1. Role hierarchy in accounting information system

In the context of accounting information system, the set of roles is

Roles = { Administrator, Manager, Accountant, Employee }.

There is inheritance relation $RH \subseteq \text{Roles} \times \text{Roles}$, written as \geq , and we have these ordered pairs:

Administrator \geq Manager, Manager \geq Accountant, Accountant \geq Employee,
Administrator \geq Employee, Manager \geq Employee, and Administrator \geq Accountant.

The role hierarchy in the accounting information system is shown on Fig. 1.

Definition 6. Let $Sat_1, Sat_2, \dots, Sat_k$ be subject attributes, $Obat_1, Obat_2, \dots, Obat_t$ are object attributes and for the subject $s, s \in \text{Subjects}$, $\text{Subject_attributes}(s) \subseteq \text{Sat}_1 \times \text{Sat}_2 \times \dots \times \text{Sat}_k$ is attribute assignment relation. Let, for the object $o \in \text{Obs}$, $\text{Object_attributes}(o) \subseteq \text{Obat}_1 \times \text{Obat}_2 \times \dots \times \text{Obat}_t$ be attribute assignment relation.

- $PF = \{f_1, f_2, \dots, f_i\}$ is a set of policy functions. Policy function decides whether subject s can access object o , when such a permission is not assigned to role. Policy function is a function, that evaluates a subset of the attributes of s and a subset of attributes of o , with a boolean value.

- For a $s \in \text{Subjects}$, $o \in \text{Obs}$ exists: $f_i(\text{Subject_attributes}(s), \text{Object_attributes}(o)) = \text{True}$. It is necessary for the access decision part of HRABAC to grant access, when the requested object does not belong to a permission, that is assigned to one of the roles of the subject, the policy function for the subject and the requested object to have True value.

In the context of accounting information system, the attributes of the subject are the fields of the database record, matching the user. If that database record is user, the subject attributes are: $Sat_1 = \text{user.id}$, $Sat_2 = \text{user.first_name}$, $Sat_3 = \text{user.last_name}$, $Sat_4 = \text{user.email}$, $Sat_5 = \text{user.telephone}$.

$\text{Subject_attributes}(s) = (\text{user.id}, \text{user.first_name}, \text{user.last_name}, \text{user.email}, \text{user.telephone})$ is an attribute assignment relation for the subject.

If the object $o \in \text{Obs}$ represents a database record salary with salary data, then the object attributes are: $Obat_1 = \text{salary.id}$, $Obat_2 = \text{salary.month}$, $Obat_3 = \text{salary.amount}$, $Obat_4 = \text{salary.user_id}$.

$\text{Object_attributes}(o) = (\text{salary.id}, \text{salary.month}, \text{salary.amount}, \text{salary.user_id})$ is attribute assignment relation for the object.

Then “own” is a policy function, $\text{own} \in PF$, $\text{own}(\text{Subject_attributes}(s), \text{Object_attributes}(o)) = \text{True}$ under the condition: $\text{user.id} = \text{salary.user_id}$. In other words, the user must “own” the salary, to view the details about it.

Definition 7. The function is_active_subject is a boolean function, $\text{is_active_subject}: \text{Subjects} \rightarrow \text{BOOLEAN}$. A subject s , for which $\text{is_active_subject}(s) = \text{False}$, does not have access rights.

Definition 8. The function is_active_role is a boolean function,
 $\text{is_active_role}: \text{Roles} \rightarrow \text{BOOLEAN}$;

a role r , for which $\text{is_active_role}(r) = \text{False}$, does not have access rights.

So, if the subject is inactive, it cannot access the requested resource. If active subject is assigned to inactive role, the access is denied via that role.

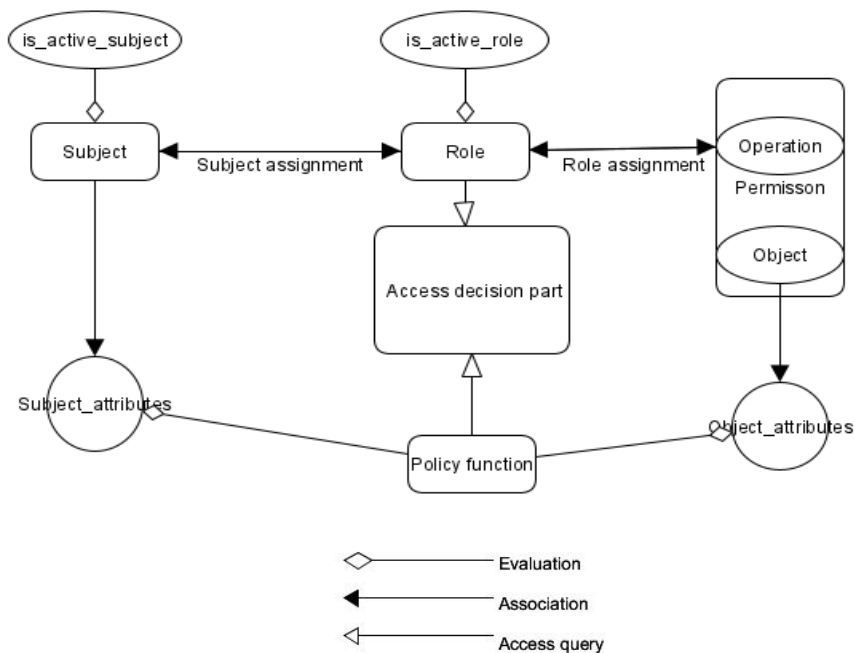


Fig. 2. The components of HRABAC

The interaction of the components of HRABAC (Fig. 2) is as it is follows.

Step 1. The function `is_active_subject` evaluates the subject, passed as argument, with a boolean value. If the evaluated value is false – go to Step 6. If the result of the evaluation is true – go to Step 2.

Step 2. HRABAC checks if there are roles assigned to the subject.

Step 2.1. If there is a role assigned to the subject: the function `is_active_role` evaluates the role with a boolean value. If that value is false – HRABAC checks for another role. If the result of the evaluation is true – go to Step 3.

Step 2.2. If there is no role, which is not evaluated – go to Step 6.

Step 3. If a permission, which contains the requested object, is assigned to the role – go to Step 5. If there is no such a permission – go to Step 4.

Step 4. HRABAC checks for policy function.

Step 4.1. If a policy function exists, which is not evaluated: HRABAC evaluates the policy function with arguments the subject attributes (`Subject_attributes`) and object attributes (`Object_attributes`), with a boolean value. If the result value is true – go to Step 5. If the result value of the evaluation is false – go to Step 4.

Step 4.2. If a policy function, which is not evaluated, does not exist – go to Step 6.

Step 5. Access Decision – the requested access is granted.

Step 6. Access Decision – the requested access is denied.

Definition 9. If s is a subject, p is a permission, a boolean function can_access is defined with arguments s and p . The function can_access is the access decision function for HRABAC. Formally, if $s \in \text{Subjects}$, $p \in \text{Permissions}$,
 $\text{can_access}: \text{Subjects} \times \text{Permissions} \rightarrow \text{BOOLEAN}$.

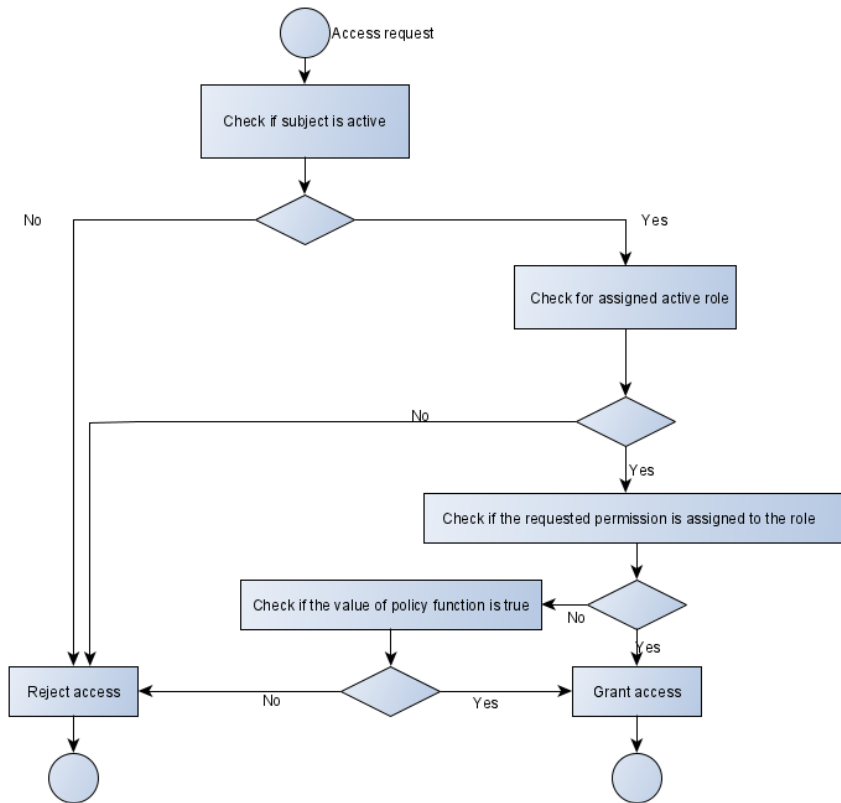


Fig. 3. The process of access control of HRABAC

The access decision function has true value, or $\text{can_access}(s, p) = \text{True}$, only when

1) $\text{is_active_subject}(s) = \text{True}$ and role r exists: $\text{is_active_role}(r) = \text{True} \wedge r \in \text{subject_roles}(s) \wedge p \in \text{assigned_permissions}(r)$;

2) $\text{is_active_subject}(s) = \text{True}$ and object $ob \in \text{Obs}$ exists, operation $op \in \text{Ops}$: $(op, ob) \in \text{Permissions} \wedge$ role r exists: $r \in \text{subject_roles}(s) \wedge \text{is_active_role}(r) = \text{True} \wedge$ policy function f exists: $f \in \text{PF}, f(\text{Subject_attributes}(s), \text{Object_attributes}(ob)) = \text{True}$.

Otherwise, $\text{can_access}(s, p) = \text{False}$.

In other words, the access decision function has true value, when the object is accessible via:

- 1) role;
- 2) policy function.

The evaluation of can_access function is shown in Fig. 3.

The scheme of access control in HRABAC is, as follows:

- First, HRABAC checks whether the subject is active.
- If the subject is inactive, the access is denied.
- If the subject is active, HRABAC checks whether an assigned active role exists. If not, access is denied.
 - For each of the assigned active roles, there is a check, whether the requested permission is assigned to role. In this is the case, access is granted.
 - If the requested permission is not assigned to role, HRABAC checks, whether the value of the existing policy functions is True to grant access. In the opposite case, the access is denied.

HRABAC can restrict access to certain actions in information systems. When it is not possible an action to be forbidden to user, HRABAC protects certain data.

The main concepts of HRABAC are:

1. Subjects and roles can be active or inactive.
2. A subject is member of a role and a permission is assigned to a role.
3. Policy functions evaluate the attributes of the subject and the object.
4. The access decision part checks whether to grant access via a role or via a policy function evaluation.

In the context of the accounting web information system, the process of evaluating access decision function `can_access` is given with pseudocode:

```
function canAccess(user, permission): boolean
begin
    if(user.is_active)
        foreach(role in user→roles)
            if(role.is_active)
                foreach(permissionIter in role→permissions)
                    if(permissionIter == permission)
                        return True;
                    endif
                endforeach
            endif
            salaryRecord =
getSalaryRecordByPermission(permission);
            if (owns(user, salaryRecord))
                return True;
            endif
        endforeach
    endif
    return False;
end
```

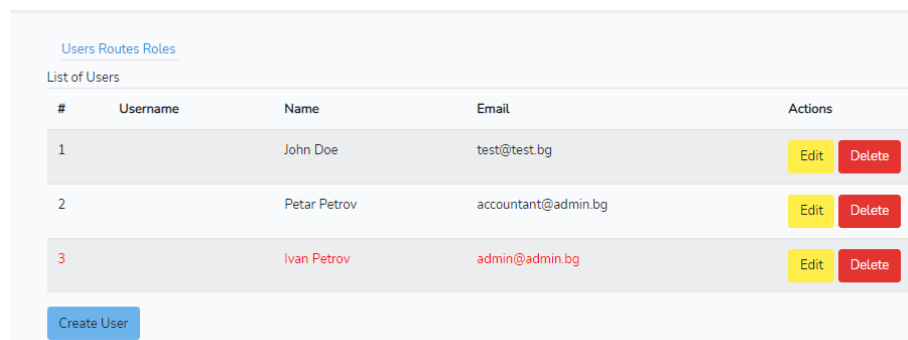
`owns` is an implementation of policy function which returns True if the user “owns” the salary.

`getSalaryRecordByPermission` returns the salary record mapping the permission. In other words, it returns the requested object.

4. Application of laravelroles/rolespermissions, that is based on HRABAC, in accounting information system

Personal data should not be shown to all people indiscriminately. It should be protected. That is one of the tasks of the software package laravelroles/rolespermissions, which is based on HRABAC, in addition to managing users, roles and permissions.

Laravel package laravelroles/rolespermission is installed on an accounting information system. This adds functionalities for managing users, roles and permissions to the accounting information system. They have graphical user interfaces. The interface “List of users” is shown on Fig. 4. The inactive users are colored in red, while the active users are colored in black. Inactive users do not have access rights.



#	Username	Name	Email	Actions
1		John Doe	test@test.bg	Edit Delete
2		Petar Petrov	accountant@admin.bg	Edit Delete
3		Ivan Petrov	admin@admin.bg	Edit Delete

[Create User](#)

Fig. 4. Graphical user interface “List of Users” of Laravel package laravelroles/rolespermissions

According to modern requirements for protecting personal data, user with role “Employee” does not have access rights to the salary data of the other users, while user with role “Accountant” manages salary data.

In other Laravel packages, that are based on RBAC, if we forbid seeing salary data for the role “Employee”, the result will be that every user with role “Employee” cannot see any salary data, including his/her own salary data.

The package laravelroles/rolespermissions provides fine-grained access control. This denotes that user with role “Employee” can see only his/her salary data in accounting information system.

The package laravelroles/rolespermissions, that is based on HRABAC, can be installed to any information system or software application, that requires managing roles and permissions, and fine-grained access control.

5. Comparison between ABAC, RBAC and HRABAC

ABAC, RBAC and HRABAC are compared by the following parameters: easy configurability, fine-grained policies, and workflow control.

Easy configurability. HRABAC is easy configurable access control model, like RBAC. That is why they are implemented in installable packages [26-29]. ABAC is

hard configurable access control model and there are no installable packages that realize that model.

Fine-grained policies. ABAC uses attributes and supports fine-grained policies. HRABAC is fine-grained too, because it can display only data for a specific user, not all data. RBAC is not fine-grained, because if a user does not have a permission to view all data, he/she cannot read the data about him/her.

Workflow control. RBAC and HRABAC can be used for workflow control. There are known permissions for each role: each job what can do. There is no data for ABAC to support workflow control.

The results of the comparison between ABAC, RBAC and HRABAC are shown on Table 1. It can be seen, that HRABAC has the advantages of both access control models RBAC and ABAC.

Table 1. Comparison between RBAC, ABAC and HRABAC

Parameter	RBAC	ABAC	HRABAC
Easy configurability	yes	no	yes
Fine-grained policies	no	yes	yes
Workflow control	yes	no	yes

The model being proposed fulfills the requirements of information systems and enterprise software of easy configuring, and at the same time, it provides fine-grained access control solution. HRABAC extends RBAC and has the advantages of RBAC and ABAC models. An exemplary accounting information system with implemented HRABAC is described in brief.

6. Future developments and conclusions

The proposed model fulfills the requirements of information systems and enterprise software of easy configuring, providing at the same time, fine-grained access control solution. HRABAC extends RBAC and has the advantages of RBAC and ABAC models. The possibility of deactivating of subject and role improves the security of the software, where the proposed model is applied.

An accounting information system is shown, whose access control requirements are impossible to be solved with RBAC or ABAC separately. The check whether the access can be granted via role or via policy function evaluation in HRABAC, provides a flexible way of access control.

The scheme of access control in HRABAC is as follows:

- First, HRABAC checks whether the subject is active.
- If the subject is inactive, the access is denied.
- If the subject is active, HRABAC checks whether an assigned active role exists. If not, access is denied.
- For the assigned active roles, there is a check, whether the requested permission is assigned to the role. In this case, access is granted.

- If the requested permission is not assigned to a role, HRABAC checks whether the value of the existing policy functions is True to grant access. In the opposite case, the access is denied.

HRABAC can restrict access to certain actions in information system. When it is not possible an action to be forbidden to user, HRABAC protects certain data. In the context of accounting information system, the attributes of the subject are the fields of the database record, matching the user. If the object represents a database record with salary data, then the user with role Employee, for which the action “read salary” is forbidden, must “own” the salary, to read the details about it. HRABAC model can be implemented in other frameworks. New models can be developed, based on HRABAC, designed for workflow control systems.

References

1. Kuhn, D. R., E. J. Coyne, T. R. Weil. Adding Attributes to Role-Based Access Control – IEEE Computer, Vol. **43**, 2010, No 6, pp. 79-81.
2. Ferraiolo, D. F., D. R. Kuhn, R. Chandramouli. Role-Based Access Control. Second Edition. Artech House, 2007.
3. Ferraiolo, D. F., R. Sandhu, S. Gavrila, D. R. Kuhn, R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. – ACM Transactions on Information and System Security, Vol. **4**, August 2001, No 3, pp. 224-274.
4. Sandhu, R., E. Coyne, H. Feinstein, C. Youman. Role-Based Access Control Models – IEEE Computer, Vol. **29**, February 1996, No 2, pp. 38-47.
5. Hu, V. C., D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, S. Karen. Guide to Attribute Based Access Control (ABAC) Definitions and Considerations – In: NIST Special Publication 800-162, SIN’13, 2014.
6. Jin, X., R. Krishnan, R. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. – In: IFIP Annual Conference on Data and Applications Security and Privacy. Vol. **7371**. Springer, 2012, pp. 41-55.
7. Frederick, G., M. Daniel, S. Sandra, G. Carol. Information Technology Control and Audit. Auerbach Publications, 2004.
8. Smith, R. E. Authentication From Passwords to Public Keys. Addison Wesley, 2002.
9. Sandhu, R., P. Samarati. Authentication, Access Control, and Audit. – ACM Comput. Surv., Vol. **28**, March 1996, No 1, pp. 241-243.
10. Schlegel, M., P. Amthor. Beyond Administration: A Modeling Scheme Supporting the Dynamic Analysis of Role-Based Access Control Policies. – In: Proc. of 17th International Joint Conference on e-Business and Telecommunications (ICETE’2020) – SECURE, 2020, pp. 431-442. ISBN: 978-989-758-446-6, ISSN 2184-7711, DOI: 10.5220/0009834304310442.
11. Guclu, M., C. Bakir, V. Hakkoymaz. A New Scalable and Expandable Access Control Model for Distributed Database Systems in Data Security – In: Hindawi, Scientific Programming. Vol. **2020**. 2020, Article ID 8875069. 10 p.
<https://doi.org/10.1155/2020/8875069>
12. Chatterjee, A., Y. Pitroda, M. Parmar. Dynamic Role-Based Access Control for Decentralized Applications – In: Blockchain – ICBC 2020. Lecture Notes in Computer Science. Vol. **12404**. Springer, Cham, 2020, pp. 185-197. DOI: 10.1007/978-3-030-59638-5_13.
13. Abdalla, M., D. Catalano, R. Gay, B. Ursu. Inner-Product Functional Encryption with Fine-Grained Access Control. – In: S. Moriai, H. Wang, Eds. Advances in Cryptology – ASIACRYPT 2020. ASIACRYPT 2020. Lecture Notes in Computer Science. Vol. **12493**. Cham., Springer, 2020, pp. 467-497.
https://doi.org/10.1007/978-3-030-64840-4_16

14. Ding, S., J. Cao, C. Li, K. Fan, H. Li. A Novel Attribute-Based Access Control Scheme Using Blockchain for IoT – In: IEEE Access, Vol. 7, 2019, pp. 38431-38441. DOI: 10.1109/ACCESS.2019.2905846.
15. Sun, S., S. Chen, R. Du. Trusted and Efficient Cross-Domain Access Control System Based on Blockchain. – Scientific Programming, Vol. 2020, 2020, Article ID 8832568. 13 p. <https://doi.org/10.1155/2020/8832568>
16. Albulayhi, K., A. Abuhussein, F. Alsubaei, F. T. Sheldon. Fine-Grained Access Control in the Era of Cloud Computing: An Analytical Review. – In: 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2020, pp. 748-755. DOI: 10.1109/CCWC47524.2020.9031179.
17. Li, H., L. Pei, D. Liao, S. Chen, M. Zhang, D. Xu. FADB: A Fine-Grained Access Control Scheme for VANET Data Based on Blockchain. – IEEE Access, Vol. 8, 2020, pp. 85190-85203. DOI: 10.1109/ACCESS.2020.2992203.
18. Meneka, M., K. Meenakshisundaram. An Enhancement Role and Attribute Based Access Control Mechanism in Big Data. – International Journal of Electrical and Computer Engineering (IJECE), Vol. 8, 2018, No 5, pp. 3187-3193. ISSN: 2088-8708, DOI: 10.11591/ijece.v8i5pp3187-3193.
19. Zigmonda, E., S. Chonga, C. Dimoulas, S. Moore. Fine-Grained Language-Based Access Control for Database-Backed Applications – The Art, Science, and Engineering of Programming, Vol. 4, 2020, No 2, Article 3. 30 p. DOI: 10.22152/programming-journal.org/2020/4/3.
20. Ding, Y., H. Sato. Bloccess: Towards Fine-Grained Access Control Using Blockchain in a Distributed Untrustworthy Environment. – In: Proc. of 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, UK, 2020, pp. 17-22. DOI: 10.1109/MobileCloud48802.2020.00011.
21. Liu, M., C. Yang, H. Li, Y. Zhang. An Efficient Attribute-Based Access Control (ABAC) Policy Retrieval Method Based on Attribute and Value Levels in Multimedia Networks. – In: Sensors 2020, Vol. 20, 2020, No 6, 1741. 15 p. <https://doi.org/10.3390/s20061741>
22. Shynu, P., K. Singh. A Comprehensive Survey and Analysis on Access Control Schemes in Cloud Environment. – Cybernetics and Information Technologies, Vol. 16, 2016, No 1, pp. 19-38.
23. Tu, S., S. Ni, M. Li. An Efficient Access Control Scheme for Cloud Environment. – Cybernetics and Information Technologies, Vol. 13, 2013, No 3, pp. 77-90.
24. Ekran Systems, 2020. <https://www.ekransystem.com/en/blog/rbac-vs-abac>
25. Penelova, M. Last Access Mart 2021. <https://packagist.org/packages/laravelroles/rolespermissions>
26. Penelova, M. Last Access Mart 2021. <https://github.com/MGP-Ucict/mpenelova>
27. Spatie. Last Access Mart 2021. <https://github.com/spatie/laravel-permission>
28. Silber, J. Last Access Mart 2021. <https://github.com/JosephSilber/bouncer>
29. Kennedy, J. Last Access Mart 2021. <https://github.com/jeremykenedy/laravel-roles>

*Received: 03.01.2021; Second Version: 29.03.2021; Third Version: 06.06.2021;
Accepted: 01.07.2021*