Applied Mechanics and Materials *Online: 2013-09-03*
ISSN: 1662-7482, Vols. 411-414, pp 35-39
doi:10.4028/www.scientific.net/AMM.411-414.35

Roles Generation for Applications in RBAC Model

Wanli Tian[1, a], LianZhong Liu[2,b], Meng Liu[3,c]

[1]Beihang University, Beijing, China

[2]Beihang University, Beijing, China

[3]Beihang University, Beijing, China

[a] buaatianwanli@gmail.com, [b] lz_liu@buaa.edu.cn, [c] 1017960816@qq.com

Keywords: RBAC; FBAC; application-oriented access control; roles generation

Abstract. RBAC has been widely used for the reason of its efficiency, convenience and safety. But as the traditional user-oriented access control strategy, the RBAC carries the disadvantage of user-oriented access control as well. It always assumes that the application is credible and the behavior of the program represents the wishes of the user. However, this assumption is increasingly proving to be false and numbers of prevalent types of security attacks leverage this weakness to misuse the authority of users. Based on RBAC and learn from the concept of FBAC, this paper will proposed a solution about generating roles for application.

Introduction

The traditional access control model, like DAC (Discretionary Access Control), MAC (Mandatory Access Control) and RBAC (Role-Based Access Control) [1] are all user-oriented access control strategy [2]. The user-oriented access control models are designed with the assumption that subjects action behalf of user, and therefor they base access control decisions on the user identity associated with each subject. However, this assumption has been proving to be false, and a number of prevent types of security attacks leverage this weakness to misuse the authority of users [3]. This is because the permissions an application needs to run are often a subset of the user's all permissions. However, with the user-oriented access control, including RBAC model, users run applications with all the permissions the user owned

Related research and works

The application-oriented access control strategy, which primarily base access decisions on the programs involved rather than on the identity users, has become a research hotpot. Unlike user-oriented access control strategy, application-oriented access control strategy constrains the behavior of the application by the identity of applications. So it can limit the threats caused by malicious software and software security vulnerabilities.

There are mainly two ways available to provide application-oriented access control strategy. One is isolation-based application-oriented access controls including sandbox and virtualization; the other is rule-based application-oriented access control [2].

The isolation-based application-oriented access control always restrict a program's permission by running an application in an environment where the application can only access objects within their sandbox. But its disadvantages are also obvious. Firstly, since all the requested resources of each application are self-contained in the sandbox, this model always needs excessive redundant resources, waste of resources and high demands on system performance; secondly, it's hard to exchange information between different sandboxes. Compared with isolation-based model, rule-based application-oriented access control can also reduce significantly safety hazards caused by malicious code, code vulnerability and design flaw. In addition, this model can also overcome the disadvantages coming with isolation-based access control model. However, rule-based access control model always has technical requirements for the user of the application.

For this reason of rule-based access control, a new model named FBAC (Functionality Based Application Confinement) [3] has been proposed. This model is inspired by the RBAC model and based on process confinement research. FBAC employs an analogous paradigm for the confinement of individual programs. While in RBAC different users share common sets of privileges relating to their roles, in FBAC different applications employs reusable and flexible policy abstraction known as functionalities [3].

FBAC has solved the problem caused by other rule-based access control, but it still exists some disadvantages:

The Functionality corresponding to role of RBAC is typically configured based on the common functions, but not all programs need only common functions. Under common circumstances, an application is always specific.

Even though FBAC supports mandatory and discretionary access control, it cannot be applied to RBAC. However, RBAC is a more effective model and used more widely.

Learning the concept of functionality from FBAC, this paper proposes a roles generation framework for applications to solve the problems that user-oriented access control models caused.

Roles generation framework

As this paper talked before, FBAC model is similar to RBAC model except that FBAC is an application-oriented access control model, but RBAC is a user-oriented access control model. This difference is made for the reason of the different subjects of them. Therefore, this paper thinks there is no need to make a new strategy from RBAC. What's need to do is to generate a role for an application in RBAC model.

The application role generation framework is shown in Figure 1.

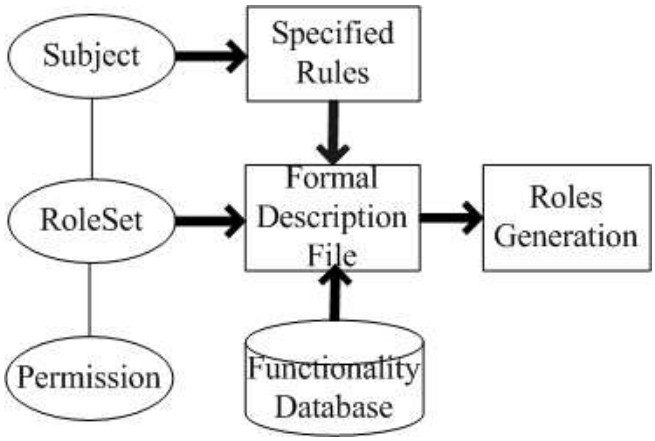

Figure 1. Roles Generation Framework

This figure just describes the part of role generation, except for the other part of original RBAC. There are three different types of policy rules:

- RoleSet, which specify the sets of roles. This set of roles is activated by one user through a session. All the permissions applications can get are subset of it.
- FunctionalitySet, which are used as modules for specifying application policies. It's similar to the concept of Functionality in FBAC.
- SpecifiedRules, which is specified by user for a specified application.

When the user is authenticated, the user will use the graphic interface to generate a formal description file. This formal description file can be specified using Backus-Naur Form (BNF) [7].

BNF is a common method for describing the format of files or information. BNF are always defined as sets of rules in the form of: nonterm ::= expression. The nonterm must be nonterminal variables and the expression is defined in terms of nonterminal variables and terminal variables. The characters "::=" is defined as the "means". A word in double quotes is on behalf of these characters. A

nonterminal or terminal in angle brackets (< >) means a necessary option. The content of the square brackets ([ ]) is optional. A '|' specifies an alternative valid value. The character '*' '+' can follow group, nonterminal variables or terminal variables. The "*" character specifies they can occur zero or more times. "+" means they can occur one or more times.

The formal description using BNF is as Table 1.

TABLE 1. FILE DESCRIPTION USING BNF

```
1.  formal_file ::= configure_rules*
2.  configure_rules ::= ws*[role_confinement | functionality_confinement |
    specified_rules]+configure_rules*
3.  role_confinement ::= "role" ws* ":" ws* role_id
4.  functionality_confinement ::= "functionality" ws* ":" ws* functionality_id
5.  specified_rules ::= "rules" ws* ":" ws* opt ws* resource_id ws* act_id
6.  role_id ::=uint
7.  functionality_id ::= uint
8.  resource_id ::= uint
9.  act_id ::= uint
10. opt ::= "del" | "add"
11. uint ::= Integer (0-65535)
12. ws ::= [space|tab|new_line]*ws*
```

This formal description file can be used directly by the access control system, but it will seriously affect the performance of the access control system. The reason is as follows:

- Different roles in Role Set and different functionalities in Functionality Set often contain a same subset of permissions, which tends to increase redundancy of database.
- Without Specified Rules, it's always necessary to get the interaction between Role Set and Functionality Set. If the Role Set and Functionality Set are present in a large number of permissions, it will cost lots of time to calculate the result. So it's very inappropriate to put this calculation onto the policy decision stage
- With Specified Rules, it will cost more system resources because the Specified Rules support more actions of set operation.

So it's necessary for future processing of the formal description file. This section assumes that we have got three sets of permission from the formal description file named RoleSet, FunctionalitySet and SpecifiedRules. Before describing the specified algorithm for generating roles of applications, it's necessary to give different levels to these three sets:

- RoleSet, which has the highest level.
- FunctionalitySet, which has the lowest level.
- SpecifiedRules's level is in the middle between RoleSet and FunctionalitySet.

The reasons, which RoleSet, FunctionalitySet and SpecifiedRules have different permission levels, are as follows:

- The permissions associated with roles activated by sessions are all the permission a user can get from this session. All the permissions applications can get are just a subset of them. Therefore, the highest privilege level is granted to RoleSet.

As we have discussed above, all Functionalities in FunctionalitySet are reusable and flexible policy abstractions. They are not customized for a specified program and the user of an application often has a better understanding of their permissions. Therefore, SpecifiedRules has a higher level than FunctionalitySet.

Roles generation algorithm

In the following algorithm of roles generation, this paper will use matrix to store permission. The matrix is familiar but different to the access control matrix [7] for each row in matrix represents one kind of resource and each column represents the same operation on different resources. This means that the same row of different columns in the matrix depict different operation to one resource.

Table 2 illustrates one example.

TABLE 2. EXAMPLE OF MATRIX

	R	W	R
Resource_1	True	False	False
Resource_2	False	False	False
Resource_3	True	True	True

The following is the required operations in algorithm of roles generation:

- getSet(RoleSet,FunctionalitySet,SpecifiedRule), which analyzes the input formal description file and stores the information to the three array referred by parameters above.
- fillMatrixWithRole(matrix,Role): This operation will transfer the permissions of Role to matrix.
- fillMatrixWithFunctionality(matrix,Functionality): This operation will traverse the permission in Functionality, then write them to matrix.

The algorithm is illustrated as Table 3:

TABLE 3. ROLES GENERATION ALGORITHM

Algorithm Name： Role Generation

Input: Formal Description File

Output: Permission Matrix of Application Role rMatrix

```
1.  Algorithm:
2.    pMatrix ← {}
3.    rMatrix ← {}
4.    getSet(RoleSet, FunctionalitySet, SpecifiedRule)
5.    for each Role ∈ RoleSet do
6.        fillMatrixWithRole(rMatrix,Role)
7.    end
8.    for each Func ∈ FunctionalitySet do
9.        fillMatrixWithFunctionality(pMatrix, Func)
10.   end
11.   for each Rule ∈ SpecifiedRule do
12.       switch Rule.opt:
13.           case “del”:
14.               pMatrix[Rule.resource_id, Rule.act_id]=false
15.               break
16.           case “add”:
17.               pMatrix[Rule.resource_id, Rule.act_id]=true
18.               break
19.       end
20.   end
21.   for each resource_id in pMatrix do
22.       for each act_id in pMatrix do
23.           If rMatrix[resource_id, act_id]== true do
```

```
24.                 continue
25.             else
26.                 pMatrix[resource_id, act_id] = false;
27.             end
28.         end
29.  end
```

Conclusion

This paper analyzes the existing problems of RBAC which is a traditional user-oriented access control. Then take the concept of Functionality of FBAC into RBAC, designing a framework of role generation. This framework brings Role, Functionality and user specified rules together, improving the accuracy of required permissions. After generating roles for applications, the subject becomes the application limited by application roles. Even if there are loopholes or malicious codes in the program, intruders will only get very limited privileges. What's more, this framework is more flexible and more consistent with the principle of least privilege than FBAC.

Acknowledgements

This work has been supported by Co-Funding Project of Beijing Municipal Education Commission under Grant No. JD100060630.

References

[1] Sandhu R S. Role-based access control[J]. Advances in computers, 1998, 46: 237-286.

[2] Cliffe Schreuders Z, McGill T, Payne C. The State of the Art of Application Restrictions and Sandboxes: A Survey of Application-oriented Access Controls and their Shortfalls[J]. Computers & Security, 2012.

[3] Dhamankar R, Dausin M, Eisenbarth M, et al. SANS[R]. Technical Report:" The Top Cyber Security Risks, 2009.

[4] Liang B, Liu H, Shi W, et al. Enforcing the principle of least privilege with a state-based privilege control model[M]//Information Security Practice and Experience. Springer Berlin Heidelberg, 2005: 109-120.

[5] Schreuders Z C, Payne C. Reusability of functionality-based application confinement policy abstractions[M]//Information and Communications Security. Springer Berlin Heidelberg, 2008: 206-221.

[6] Knuth D E. Backus normal form vs. backus naur form[J]. Communications of the ACM, 1964, 7(12): 735-736.

[7] Sandhu R S. The typed access matrix model[C]//Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on. IEEE, 1992: 122-136.

[8] Schreuders Z C, Payne C. Functionality-based application confinement: parameterised hierarchical application restrictions[J]. Proceedings of SECRYPT, 2008: 72-77.