International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 10, October - 2013

Enhancement of Java Access Control using Spring with AOP and AspectJ

K. Vinod Kumar[1], Dr. A. P. Siva Kumar[2], K.Thyagarajan[3]
[1] Department of CSE, JNTUA, India, cse.
[2]Assistant Professor, Department of CSE, JNTUA, Anantapur, India,
[3]Associate Professor, Department of CSE, SVCET, Chittoor, India,

Abstract

In java applications, access control is used to prevent the resources from unauthorized access but it is difficult to provide security for cross cutting concerns. Among the techniques introduced to address this issue, AOP stands out to be the best; it being strong in terms of reducing the code scattering and tangling. However, this faces some challenges in case of dynamic control for exchanging the policies and reusability. To overcome this we present one framework called AOJAC (Aspect Oriented Java Access Control) which supports access control policies which use different kinds of context information and enables the change of these policies at runtime.

1. Introduction

A key principle in software engineering, proposed by Dijkstra named separation of concerns. This principle is applied when a complex problem with different concerns is properly identified, address separately and finally the respective solutions are integrated to produce the final result there by used for controlling the complexity of the application.

This principle has been followed by access control system architecture which is traditionally based on a model named abstract reference monitor proposed by Anderson [].Obviously it separates the security logic (access control logic) from the main logic of the application. This model implementation has been difficult, because access control is a crosscutting concern i.e., security requirement that crosscuts the application requirement.

In order to separate these crosscutting concerns a new methodology AOP is introduced. The figure 1 shows the process of AOP implementation in java. It introduces a new modularization unit called aspects which crosscut the modules. The aspect weaver is the process of producing the final system by integrating the core and crosscutting modules.

The AOP was implemented in a language called AspectJ. This method is well suited for providing the security in the case access control system. Access control is the process of preventing the resources from unauthorized access. However, current access control solutions using these techniques have typically not been reusable or generic.

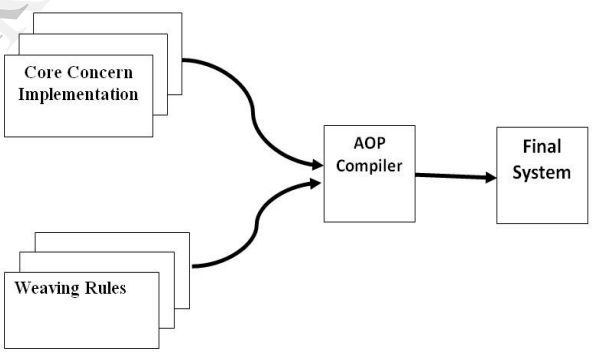

Figure 1: Java based AOP implementation

To overcome this issue we propose the framework called AOJAC (Aspect Oriented Java Access Control) using spring framework and AspectJ language. It is especially designed for the java application. The framework allowing it to address some of the problems found in the application of industry standards such as JAAS (Java Authentication and Authorization Service). It also uses the concepts of AOP to separate the crosscutting concern and java5 annotations to specify the application's protected objects and their access control requirements.

A few important AOP notions are:

- Join point-A well identifiable place in the execution of the program. For example, calling a method or assigning a value to a member in an object. Since

they are the places where the crosscutting actions are taken in.

- Pointcut – it is a program construct which selects join points and collects the information at those points.
- Advice – execution of code at join points selected by the pointcut.

The remainder of this paper is structured as follows. Section II presents the access control models and shows the design of access control architectures. Section III describes the demerits of JAAS when added to user-centric access control model in Java language. Section IV introduces the AOJAC framework concepts and an algorithm which describes how the protected object is being accessed by unauthorized people. Section V presents a case study to demonstrate the proposed framework while section VI concludes the paper.

2. Access Control

Access control, which assures the protection of resource against unauthorized access, is a security service. It includes the concepts of access control models and representation of access control architecture.

2.1. Concepts

The definition of an access control policy is indicated by the development of an access control system and its enforcement through appropriate security mechanisms. Access control models [],[] represent formally the access control policies, expressed through specific access control languages like Ponder, Security Policy Language (SPL) and extensible Access Control Markup Language (XACML).

Generally, there are two access control models:

i. Mandatory access Control (MAC) model: This model used in systems were rigorous access control is very important are the one in which access rules can change over time, system-wide, usually fixed and hence users cannot influence them.
ii. Discretionary access control (DAC) model: It gives the owner of the protected object and the permission needed to access the right in determines the access control policy for that object. This system has the ability to act or decide according to the user judgment and is capable of granting access to that protected object to other users.

Generally, a discretionary access control policy stated by a set of authorizations in turn defined as a tuple (s,m,o,pred) defines that subject may legitimately use the access mode m which represent a specific operation performed over the object or an abstract access mode used to access the protected object o if the predicate pred is true in the context of the access related to a set of specific operations. Instead of simple one, the access mode has the advantage to decrease the number of permissions within the system as it is associated to a set of operations.

The use of predicates augments the expressiveness of the authorizations, supporting a more fine-grained control of authorizations.

RBAC was proposed a model basing on the MAC and DAC models. This model was well received, since the notion of roles fits well to the common notion of function in organizations. RBAC models associate authorizations to roles performed by subjects. RBAC authorization (r, m, o, pred) states that a subject performing a role can legally allow accessing the protected object based on the given mode whenever the predicate is true. Since subjects are not directly associated with access modes, but indirectly through the role or roles they perform, the management of individual privileges in the system is often only a matter of assigning the appropriate roles to each subject.

2.2. Access Control System Architecture

Access control system architectures which traditionally bases on the abstract reference monitor proposed by Anderson [], intercepts all access attempts from subjects to the protected objects. Conceptually, a reference monitor has two main functions:

- A decider, responsible for evaluating the legitimacy of the accesses,
- An enforcer, responsible for intercepting all access attempts and enforces the decision that was taken.

According to this model, all access attempts are intercepted by the enforcer, which asks the decider to determine the legitimacy of the access, searching the authorizations database.

To avoid scattering enforcer code throughout the application, many techniques addressing the problem of scattered concerns have been used, such as design patterns, particularly the proxy pattern, meta-level or reflexive architectures, and, more recently, AOP, in which our work fits.

3. Access Control in JAVA

The access control model is used to prevent the resources like files from unauthorized access to the Java application. It is a code centric model because the subject is defined within this model according to the code origin. Once JAAS was added to the J2SE (Java 2 Platform Standard Edition) Development kit (JDK), and hence its core became a part with version 1.4. JAAS added a new access control model

to the Java language called user-centric model. Besides users can be subjects, protected objects can also be specific application resources, i.e., specific application functionalities.

In JAAs, class subject is used to represent a user authenticated in a given system. A subject being an aggregation of principal represents different entities which derives its authority form the subject. JAAS supports RBAC model, suppose for e.g., a principal may be a username, the name of a group to which she is associated, or a role that she performs.

JAAS does not support the features like allowance of making changes to access control policies during run time, predicates which limits the expressiveness of its authorization and finally does not allow the separation between the implementation code and application logic. While trying to protect specific objects, in the place of access control an enforcer function should be placed explicitly in the application.

4. AOJAC framework

In this section we introduce AOJAC, an access control framework for Java applications that uses the abstract reference monitor suggested by Anderson. It is reusable and supports authorizations with domain specific information.

4.1. Access Control Model

Basically an authorization is defined as a tuple (s, m, o, pred) where in s is subject, m is access mode, o is protected object and pred is predicate.

If predicate is true based on the access mode, subject or user allows accessing the protected object and thus helps to meet the requirements of access control for java application. Conceptually, the entities of tuple are described below:

- Subject- the framework is not restricted to particular kind of subjects, i.e., subjects may be users, groups or roles, etc
- Access modes: abstract access mode grants the permission to the subjects which perform their related operations over the protected objects. This process is followed by access control model.
- Protected Objects: the framework aims to protect the data members and member functions in an object. The operations performed over the data members are either set or get the values while in member function the method is to be executed.
- Predicate- basically, predicates are used to increase the authorizations expressiveness, defining fine-grained permissions and providing more restriction to their application.

  This can be used in different ways, such as:
  - User characteristics: name, date of birth, gender and nationality of a user etc.
  - Object characteristics
  - Some other external conditions: scope and localization of access, relation between entities.

Basing on this, authorization is classified into two tuples: (s, m, pred) and (m, op, po). The first tuple defines the subject has access mode only if the predicate is true and hence called authorization. The second tuple defines if the subject has access mode, it allows performing operations on protected object and hence called access control requirement.

4.2. Algorithm

Following steps describes how protected objects is being accessed by authorized people

1. Whenever the subject attempts an access over protected object, immediately the enforcer intercepts the access.
2. The enforcer collects the information, particularly the one the decider wants i.e.,
   a. The subject
   b. The protected object and its usage context like the method, the arguments related to the invocation of the method.
   c. The abstract access modes required to access the protected object and other architectural meta-information that the enforcer may need.
3. The decider collects the required information to evaluate whether the access is legal or not.
4. In order to check the status of access legitimacy, the decider search in access control policy which are placed externally from the code (e.g., in JAAS policy files or XACML)
5. The status of step 4 is returned to the enforcer.
6. Based on the decider status, the enforcer takes in the corresponding action which are prescribed below:
   a. If the access is legal, the access proceeds.
   b. An exception is thrown in the case of access is denied.

5. Case Study: A Web Application

To demonstrate AOJAC framework let's take a small web application. Here users are the controllers. User can create the notes, important things can be written there. By default they have their own permission to their notes. If any dynamic changes are needed, change in the code is mandatory. Conversely, in the first case of unknown number of users an aspect method can be used. For example, when a user wants write permission to his note

dynamically; an aspect gets invoked automatically then he could be able to access that resource (note) in the write mode. Our application runs on few permissions and they are: NO, READ, WRITE, MANAGE and AUTH.

Permission	Description
NONE	No access.
READ	The user can only read the entries in the note but cannot modify, create or delete entries.
WRITE	Users can read and write entries in their notes.
MANAGE	User can read and write entries in his note. Additionally it includes deleting a "note".
AUTH	User can read, write, and delete a note and additionally manage permissions for a note

In the case of known number of users, we will know where the permission exchanges occurred in the code so we can pre-plan the code. But in the case of new entry in the application, it is difficult to identify where the permission exchanges takes place. This will increase code complexity especially when we deal with the raw code.

If it is pre-planned ,there is no problem in case of complexity (that is if number of customers we have is known, then who issues the permission and who requests that permission will be known).But whenever a new user is generated then the code complexity will be increased because we don't know which user issues permission to which user. To overcome this issue dynamic control is needed. Suppose, if a user wants to assign the permission to another user to his resource during execution time without making any changes in the code; this is possible by using an aspect model which uses annotations and hence based on given condition it is invoked automatically.

Consider an example for managing notebook in an application, it allows users to add and remove their personal note and we can add important things in their notebook. Additionally it avails users to share single notes with other users. If any user needs any resource (note) to access, we need to check whether that user has the authority to access that resource. This could be done by using access control mechanism. To perform access control permissions automatically, we can place AOJAC annotations in their relevant place in the source code.

These annotations can be controlled by managers, they are:

- Access manager: it acts like an interface between the authorization decisions and the authorization providers. For each and every permission check, we can add @secure and @filter annotations and its related methods are defined on access manager interface. The object itself contains instance methods for creating and deleting notebook entries. For instance-level permission check we can use @secure and @filter annotation.
  - @secure annotation: It can be applied at argument level as well as method level. In parameter level, whenever an object is passed as an argument during invocation of method, this annotation makes the permission check. In method level, whenever corresponding method is invoked this annotation checks the permission.
  - @filter annotation: This checks if a requestor has read access to objects returning from method invocation else corresponding object is removed from the result. This can also be applied on multiple objects like arrays and collections.

- Authentication manager: this manager controls whenever an unauthorized user tries to access the resource. It includes @Authorizationserviceprovider is annotated with access manager interface.
- Crypto manager: @Encrypt annotations are recognized by the AOJAC when they are kept on the member of objects which are annotated with @secure.

By using these managers, we can control the policies during the execution time.

6. CONCLUSION

This proposed framework inspired by a proposal of Laddad [1] is used to modularize JAAS (Java Authentication and Authorization Service) client code using AOP. The framework was implemented in the AOP Language called AspectJ. The framework consists of two layers, which omits the problems like reducing the visibility of aspects, avoiding the pointcut and advice loops because in the first layer controlling the spring transaction takes place and in the second layer which dynamically exchanging the policies between the multiple users takes place. The proposed method is mainly focused on getting control over dynamic control during runtime. It also incorporated the AOP concepts which add the advantage of separation the crosscutting concerns. We built a small web application to demonstrate the process of dynamic control during the execution time.

REFERENCES:

[1] Ramnivas Laddad. *AspectJ in Action*. Manning, Greenwich, Connecticut, 2003.(AspectJ in action)

[2] Anderson, J. P., *Computer Security Technology Planning Study*. Technical Report ESDTR- 73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. (Also available as Vol. I, DITCAD-758206. Vol. II DITCAD-772806).- abstract reference monitor

[3] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. ACM Transactions on Database Systems, 23(3):231–285, September 1998.

[4] G. Ahn and R. Sandhu. The RSL99 language for role-based separation of duty constraints. In Proc. of the fourth ACM Workshop on Role-based Access Control, pages 43–54, Fairfax, VA, USA, October 1999. –(2,3 access control models)

[5] McGraw, Gary. *Software Security: Building Security In*. Boston, MA: Addison-Wesley Professional, 2006 (ISBN 0-321-35670-5).(security software engineering)

[6] Baumeister, Z. and K. Knapp, "Aspect-Oriented Modeling of Access Control in Web Applications". In *Workshop on Aspect Oriented Modeling* (AOM'05), 2005.

[7] R. Bodkin, "Enterprise Security Aspects", In *AOSD Tech. for Application-Level Security* (AOSDSEC'04), 2004.

[8] Chargi, A., and M. Mezini, "Using Aspects for Security Engineering of Web Service Compositions", In *Proc. IEEE Int'l Conf. on Web Services* (ICWS'05), pp. 59-66, 2005.

[9] Devanabu, P. and S. Stubblebine, "Software Engineering for Security: A Roadmap", In *Proc. Conf. Future of Software Eng., ICSE'00, Special Volume*, pp. 227-239, 2000.

[10]Kiczales, G. et. al., "Aspect-Oriented Programming", In *Proc. European Conference on Object-Oriented Programming* (ECOOP'97), 1997.

[11]Kiczales, G. et. al., "Overview of AspectJ", In *Proc. European Conference on Object-Oriented Programming* (ECOOP'01), 2001.

[12]Rosenhainer, L., "Identifying Crosscutting Concerns in Requirements Specifications", In *Early Aspects 2004: Aspect-Oriented Requirements Eng. and Architecture Design Workshop*, pp. 49-58, 2004.

[13]Ray, I., France, R., Li, N. and G. Georg, "An Aspect- Based Approach to Modeling Access Control Concerns", *Journal of Info. and Software Tech.*, 46(9), July 2004, pages 575-587.

[14]Viega, J., Bloch, J.T., and P. Chandra, "Applying Aspect-Oriented Programming to Security", In *Cutter IT Journal*, 14(2):31-31, 2001.

[15]B. De Win, B. Vanhaute, and B. De Decker. How aspect-oriented programming can help to build secure software. *Informatica (Ljubl.)*, 2002.