IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 227-231

Modularization of Enterprise Application Security Through Spring AOP

Kotrappa Sirbi[1] & Prakash Jayanth Kulkarni[2]

[1]Department of Computer Science & Engineering, K L E's College of Engineering & Technology, Belgaum, India
[2]Department of Computer Science & Engineering, Walchand College of Engineering, Sangli, India
Email: [1]kotrappa06@gmail.com, [2]pjk_walchand@rediffmail.com

ABSTRACT

The goal of the paper is to present that Aspect Oriented Programming AspectJ integrated with Spring AOP provides very powerful mechanisms for modularizing enterprise security cross-cutting concerns. Aspect-oriented programming (AOP) allows weaving a security aspect into an application providing additional security functionality or introducing completely new security mechanisms.Spring's proxy-based AOP framework is well suited for handling many generic middleware and application-specific problems. The most important part of the Spring/AspectJ integration allows Spring to configure AspectJ aspects using Dependency Injection. This brings similar benefits to aspects as to objects. Also Spring has ability to provide AspectJ pointcut expressions to be used in Spring XML or other bean definition files, to target Spring advice. This will allow some of the power of the AspectJ pointcut model to be applied to Spring's proxy-based AOP framework. Implementation of security with AOP is a flexible method to develop separated, extensible and reusable pieces of code called aspects. In this inquisitive study, we bring some insight about the usage of powerful technology for developing secure enterprise applications.

Keywords: Aspect Oriented Programming (AOP), Object Oriented Programming (OOP), AspectJ, Spring AOP, Plain Old Java Object (POJO), Acegi

1. INTRODUCTION

The most fundamental principle in software engineering is that the separation of concerns leads to a system that is simpler to understand and easier to maintain [13]. Various methodologies and frameworks support this principle in some form. For instance, with OOP, by separating interfaces from their implementation and it helps to modularize the core concerns well. But for crosscutting concerns, OOP forces the core modules to embed the crosscutting concern's logic. Although the crosscutting concerns are independent of each other, using OOP leads to an implementation that no longer preserves independence in the implementation.OOP technology to solve these problems often causes system difficult to understand, achieve and maintain. AOP is a new programming technology, which allows developers to focus on cross-cutting concerns. Aspect-oriented Programming (AOP) provides the abstraction in concern level by the horizontal mode, and it is a new programming model, which make the modularizing crosscutting concerns as aim in higher abstraction level. AOP is real because it's been widely used and adopted in enterprise applications, Web and Application servers, Application frameworks, Monitoring tools, Compilers and IDE Integration etc., [13].

The rest of the paper is as follows: section 2 an introduction AOP enterprise security, section 3 discuss about concept of Spring AOP and AspectJ concepts. The section 4 describes about spring AOP security features and section 5 Spring AOP pre-built solutions and section explain the methodology of implementation of Spring AOP security cross-cutting concern and section 6 contains conclusion of the paper.

2. AOP IN ENTERPRISE APPLICATION SECURITY

Enterprise applications need to address many crosscutting functionalities: transaction management, security, auditing, service-level agreement, monitoring, concurrency control, improving application availability, error handling, and so on [1]. Many enterprise applications use AOP to implement these functionalities. All the examples in given here are based on real-world problems and their AOP solutions. Virtually every project that uses Spring uses AOP [9]. Many applications start with prewritten aspects supplied with Spring (primarily transaction management and security). But due to the AspectJ syntax, writing custom aspects is becoming a common task. After reaching the limits of Spring AOP, many applications move toward AspectJ weaving. The typical trigger point for this change is crosscutting of domain objects or other forms of deeper crosscutting functionalities. At that time, it's common to start with the AspectJ syntax (which is used with Spring's proxybased AOP) along with the load-time weaver. Of course, applications that don't use Spring often use AspectJ weaving from the beginning [9].

The IT Industry sectors in which AspectJ is used in production (with proxy-based and bytecode-based AOP)

range from financial companies (banking, trading, hedge funds) and health care to various web sites (e-commerce, customer care, content providers, and so on).The common notion is that if company implementing enterprise applications and using Spring AOP, it is a good company.

3. ASPECT ORIENTED PROGRAMMING (AOP) AND ASPECTJ

3.1. AOP Concepts

Aspect-Oriented Programming (AOP) is the invention of a programming paradigm developed by the Xerox Palo Alto Research Center (Xerox PARC) in the 20th century [6, 8]. It allows developers to separate tasks that should not be entangled with the cross-cutting concerns, so as to provide better procedures for the encapsulation and interoperability. The core thought of AOP is to makes a complex system as combinations by a number of concerns to achieve. After demand researched, the concerns are divided into two parts: cross-cutting concerns and core business concerns. Core business concern is the needs of the business logic, business various subsystems, such as financial systems, personnel systems. And cross-cutting concern is the needs of the various subsystems business, may be involved in some of the public functions, such as log records, security and so on.

3.2. Spring and AOP and AspectJ

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as security that cut across multiple types and objects [8, 10, 13, 15, 16].

AOP concepts are:

i. *Aspect:* A modularized implementation of a software concern that cuts across various objects in a software implementation. Logging is a good example of an aspect. In Spring AOP, aspects are nothing more than regular Spring beans, which themselves are plain-old Java objects (POJO) registered suitably with the Spring Inversion of Control container. The core advantage in using Spring AOP is its ability to realize the aspect as a plain Java class.

ii. *Join Point:* A point during program execution, such as a method executing or an exception being handled. In Spring AOP, a join point exclusively pertains to method execution only, which could be viewed as a limitation of Spring AOP. However, in reality, it is enough to handle most common cases of implementing crosscutting concerns.

iii. *Advice:* Information about "when" an aspect is to be executed with respect to the join point. Examples of types of advice are "around," "before," and "after." They specify when the aspect's code will execute with respect to a particular join point.

iv. *Pointcut:* A declarative condition that identifies the join points for consideration during an execution run. Pointcut is specified as an expression. Spring AOP uses the AspectJ pointcut expression syntax. An example pointcut expression is: execution(* com.myorg.springaop. examples.MyService*.*(..)). Asterisks in the expression refer to wildcards, as is conventional.

v. *Introduction:* (Also known as an inter-type declaration). Declaring additional methods or fields on behalf of a type. Spring AOP allows introducing new interfaces (and a corresponding implementation) to any proxied object.

vi. *Target Object:* Object being advised by one or more aspects. Also referred to as the advised object. Since Spring AOP is implemented using runtime proxies, this object will always be a proxied object.

vii. *AOP Proxy:* An object created by the AOP framework in order to implement the aspect contracts (advise method executions and so on). In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy. Proxy creation is transparent to users of the schema-based and @AspectJ styles of aspect declaration introduced in Spring.

viii. *Weaving:* Linking aspects with other application types or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime. The various steps used in the present face recognition system are discussed below.

4. SPRING AOP SECURITY FEATURES

Spring security along with AOP can simplify the implementation of the classic crosscutting concern of securing enterprise application [11, 13]. The security aspect needs to do two things, firstly select join points that need authentication or authorization and then advise the selected join points to perform authentication and authorization(shown in lising1).

```
aspect SecurityAspect {
private AccessDecisionManager accessManager;
pointcut secured() : ...

before() : secured() {
SecurityAttribute sa = ...
accessManager.checkPermission(sa);
}
}
```

Lising1: Basic code for Aspect Security

The interesting piece of code is the computation of the security attributes. In the conventional technique, each method creates a separate security attribute object. In an AOP solution, the same advice applies to all advised join points, yet each join point may require a different attribute. Therefore, the advice may need some cooperation from the advised code or some external configuration to compute a correct attribute at each join point. One way to achieve this collaboration is to use annotations and also security aspects can be applied using either the proxy-based or byte-code based AOP. The security aspect acts as a controller that mediates between the core system and the security subsystem.

Security requirements vary widely, and many implementations exist to meet these needs [2, 3, 7, 9]. For example, an authentication requirement may vary from simple web-based authentication to a single sign-on solution. Storage for credentials (such as passwords) and authorities (typically roles such as ADMIN or USER) varies widely as well from a simple text file to a database or Lightweight Directory Access Protocol (LDAP). These variations make the already complex topic of security even more so. The increased complexity warrants raising the level of abstraction. But creating such an abstraction is very complex task. This is where Spring Security comes into play. By providing an abstraction layer and an implementation for most commonly used security systems. Furthermore, Spring Security provides ready-made solutions for a few common scenarios that allow implementing certain security requirements by including just a few lines of configuration. Security is an important consideration in modern, highly connected software systems.

Most applications need to expose functionality through multiple interfaces to allow access to the business data and make complex integration possible. But they need to do so in a secured manner. It isn't a surprise that most enterprises spend substantial time, energy, and money to secure applications. Security consists of many components such as authentication, authorization, auditing, protection against web site attacks, and cryptography. This paper shows how Spring Security up close to reality through implementation of authentication and authorization.

4.1. Authentication

Authentication is a process that verifies that the user (human or machine) is indeed whom they claim to be. For example, the system may challenge the user with a username and password. When the user enters that information, the system verifies it against the stored credentials. Spring Security supports authentication using a wide range of authentication schemes such as basic, forms, database, LDAP, JAAS, and single sign-on. It can also be roll its own authentication to support the specific scheme that the organization is using (and still utilize the rest of the framework, including authorization). After the user credentials have been authenticated, the authenticated user (more generally known as the principal) is stored in the security context. Figure 1 show the overall structure used in Spring Security for authentication.

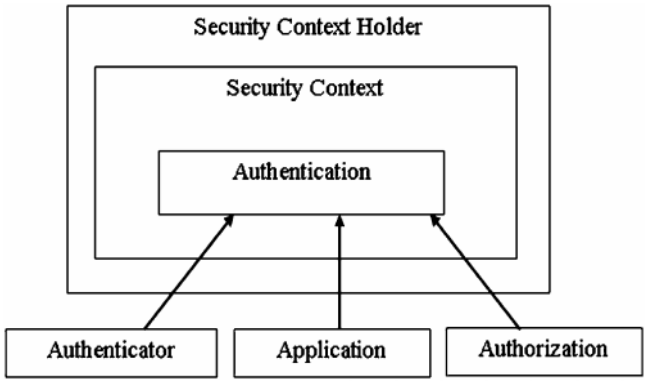

Fig. 1: Authentication in Spring Security

4.2. Authorization

Authorization is a process that establishes whether an authenticated user has sufficient privileges to access certain resources. For example, only users with the admin privilege may access certain web pages or invoke certain business methods. Spring Security provides role-based and object-level authorization. To accommodate potentially complex custom requirements, it provides several components that can customize. Figure 2 depicts the authorization sequence.

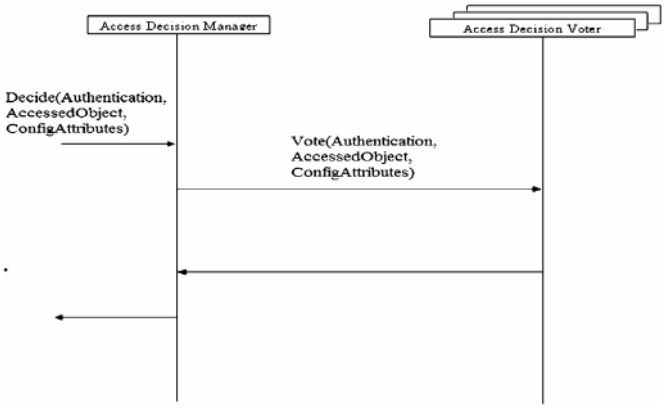

Fig. 2: Spring Security Authorization Sequence

5. AOP PROXY FRAMEWORK-IMPLEMENTING AUTHORIZATION ASPECTS

To implementing security using proxy based Spring AOP involves the arrangement (shown in figure 3) .Spring AOP creates a proxy around the service beans, and the security advice ensures authorized access. Spring AOP works only with Spring beans. If there is requirement of more encompassing solution, use of byte-code weaving is the better often. In the same way as the authentication aspect, the mechanism to choose authorized join points and to obtain security attributes varies. A simple example of security cross-cutting concerns implementation is considered here to demonstrate the power of Spring AOP.To implement security cross-cutting concern, specifically authorization, we will use Acegi. Acegi is a popular and flexible security framework that can be used in enterprise applications.

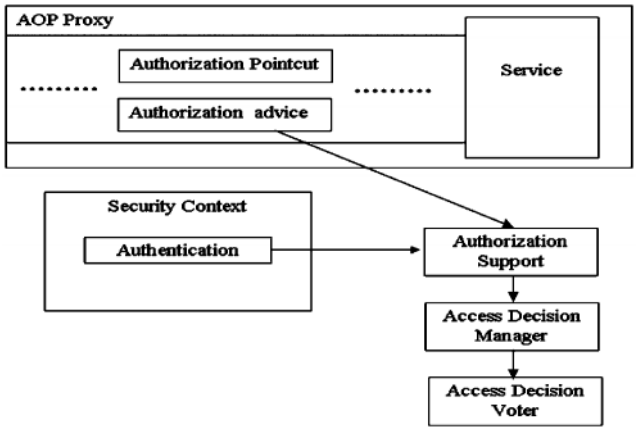

Fig. 3: Securing the Service Layer using Proxy-based AOP and Spring Security

Acegi integrates well with Spring and uses Spring application contexts for all configurations. Using Acegi Security greatly simplifies implementing authorization in this application in a flexible manner. To start, a POJO class that will implement the security concern as a Spring AOP aspect. Then the as shown in listing 2, the implementation of method *checkSecurity(…)* involves using Acegi APIs and classes such as *SecurityContext Holder* for retrieving the authenticated user and the user's role. Using the user and his role, an application-specific authorization can be implemented:

```
import org.acegisecurity.context.SecurityContextHolder;
import org.acegisecurity.userdetails.UserDetails;
import org.aspectj.lang.ProceedingJoinPoint;

public class MySecurityAspect
{
   public Object checkSecurity(ProceedingJoinPoint call) throws Throwable
   {
      System.out.println("from security aspect: checking method call for "
   + call.toShortString());

   Object obj = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

      String username = "";
      if(obj instanceof UserDetails)
        username = ((UserDetails)obj).getUsername();
      else
        username = obj.toString();

      //Do authorization check here

      System.out.println("from security aspect: authenticated user is "+username);
      return call.proceed();
   }

}
```

Listing 2: Spring AOP Aspect

The important thing to note is the entire authorization check lies in a separate aspect (POJO class), which is distinct from the business logic code. This security aspect can be effectively applied to our business logic method using the following Spring configuration: First, register the regular Java class *SecurityAspect* with Spring as a Spring bean and then specify the pointcut and advice. The pointcut expression is execution *(*com.myorg.springaop.examples.MyService*.*(..))* and the advice type is "around," and the aspect method name is *checkSecurity.*

The Spring configuration for the security aspect (as shown in listing 3):

```
<bean id="SecurityAspect" class="com.myorg.springaop.examples.MySecurityAspect"/>

<aop:config>
    <aop:aspect ref="SecurityAspect">
      <aop:pointcut id="myCutSecurity"
           expression="execution(* com.myorg.springaop.examples.MyService*.*(..))"/>
      <aop:around pointcut-ref="myCutSecurity" method="checkSecurity"/>
    </aop:aspect>

    ...
    ...
    ...
</aop:config>
```

Listing 3: Spring Configuration for the Security Aspect

Additionally, a Spring configuration is needed for configuring Acegi with Spring. The configuration uses an in-memory data access object (DAO) provider, in which case, the developer specifies the potential users and roles in the application as simple name-value pairs, as a part of the static Spring configuration file. This can easily be inferred from the following Spring configuration (as shown in listing 4):

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="daoAuthenticationProvider"/>
    </list>
  </property>
</bean>

<bean id="daoAuthenticationProvider"
  class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">
  <property name="userDetailsService"><ref bean="inMemoryDaoImpl"/></property>
</bean>

<bean id="inMemoryDaoImpl"
class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      shiva=mypassword,ROLE_TELLER
      paru=mypassword,disabled,ROLE_TELLER
    </value>
  </property>
</bean>
```

Listing 4: Spring Configuration File

5.1. Spring Security Prebuilt Solutions

Spring Security provides ready-made solutions that enable developers to secure applications with a few lines of configuration. These solutions target different parts of application: web, service layer, and domain objects.

5.1.1. Web Security

Securing web applications is a common task, therefore, Spring Security provides special support for this scenario. With namespace-based configuration, a few lines can configure URL level security that ensures that

the user has the right authority to access the URLs. For example, the following Spring configuration provides Authentication using a default login page and ensures that any URLs that end with "delete.htm" are accessible only by users with the ADMIN role. Other URLs are accessed only by users with the USER role.

```
<security:http auto-config="true">
<security:intercept-url pattern="/*delete.htm" access="ROLE_ADMIN"/>
<security:intercept-url pattern="/**" access="ROLE_USER"/>
        </security:http>
```

Listing 5: User Authentication

The declaration of user service is to provide username/password. Typically, firstly start with a simple snippet as shown listing 5, modify various attributes, and add additional elements to tailor to the specific needs.

5.1.2. Service Level Security

Because most enterprise applications utilize a service layer as the exclusive way of accessing business functionality, it makes sense to secure this layer. Spring Security provides prebuilt aspects along with namespace-based configuration support to secure the service layer with minimal code. It offers two options to specify the access-control information: through the XML-based configuration and through annotations.

6. CONCLUSION

Spring Security along with AOP can simplify the implementation of the classic crosscutting concern in modularizing security cross-cutting concern for enterprise application. We have shown that both Spring AOP and AspectJ strive to provide comprehensive AOP solutions and complement each other. For the future work AOP and Spring technology can make big difference for software developer implementing security concern, particularly for secure enterprises application development AOP research, can develop more space, not be limited to the code level.

REFERENCES

[1] Xin Ma, and Lian-he Yang, "The AOP Research based on Enterprise Application", *World Academy of Science, Engineering and Technology*, **45**, 2008.

[2] Anderson, "J. P. Computer Security Technology Planning Study", Tech. Rep., ESD-TR- 73-51, Oct. 1972.

[3] Welch, I. S., and Stroud, and *R. J. Re-engineering Security as a Crosscutting Concern. Comput. J 46*, *5*, 2003, 558–589.

[4] AOP Issue, Communications of the ACM, **44**, No. 10, 2001 [5] Elrad, T, R. Filman, A. Bader. Aspect-Oriented Programming. CACM, **44**, Number 10, 2001.

[6] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. "An Overview of Aspect J. In ECOOP 2001" (Budapest, Hungary, 2001), vol. LNCS 2052, Springer-Verlag, pp. 325–353.

[7] GASSER, *M. Building a Secure Computer System*, Van Nostrand Reinhold Co., New York, NY, USA, 1988.

[8] AspectJ Group Tutorial - Aspect-Oriented Programming with AspectJ, AspectJ Workshop, Xerox PARC, 2002 http://www.parc.xerox.com/groups/csl/projects/aspectj/

[9] Aspect Oriented Programming with Spring; Spring Framework: http://www.springframework.org/docs/reference/aop.html

[10] AspectJ. Available from http://www.eclipse.org/aspectj/

[11] http://www.springsource.org/

[12] http://www.acegisecurity.org/

[13] Laddad, R. Enterprise AOP with Spring Applications: AspectJ in Action 2nd Edition, Manning Publications, 2010.

[14] Gollmann, *D. Computer Security*, John Wiley and Sons, 2002.

[15] Laddad, R. AspectJ in Action: Practical Aspect-Oriented Programming, Manning, 2003.

[16] Gradecki, J. D., and Lesiecki, *N. Mastering AspectJ: Aspect Oriented Programming in Java*, John Wiley and Sons, 2003.