Toward a Reusable and Generic Security Aspect Library

Minhuan Huang, Chunlei Wang, Lufeng Zhang

Beijing Institute of System Engineering
P.O.B 9702-19, 100101 Beijing, China
{hmh, wcl, zlf@public.bise.ac.cn}

Abstract: Security is a good example for Aspect-Oriented Programming, but there are few reusable software components at the level of aspects. We introduce an elementary implementation of a reusable and generic aspect library providing security functions. This aspect library is based on AspectJ and common Java security packages, and includes typical security mechanisms. We describe the principle, architecture and usage of the security aspect library, and give a practical example of application in which security has been implemented using the aspects in the library. We also discuss the advantages and disadvantages of aspect library in reusability and generality, and the future efforts we will focus on.

Key Words: AOP, application-level security, aspect reusing, aspect library.

1 Introduction

Software security usually can be viewed as the combination of two critical factors: security mechanisms and tools constructed from various security technologies (e.g. encryption/decryption, authentication and authorization, etc), and applying these technologies to software in a globally consistent way. In fact, the modularization of security-related functions, such as access controlling, or defensive input checking, is hardly to achieve with nowadays mainstream software engineering technologies [1]. Also as mentioned in [2], software security usually has the so called “cross-cutting” nature where a solution to a single problem might involve modifications across the application. Aspect-Oriented Programming [3] has been proposed as a promising approach for modularizing cross-cutting concerns in software. AOP provides another dimension of modularity, “aspect”.

In this paper we describe an elementary implementation of a security aspect library which provides reusable and generic security-relevant aspects for constructing secure applications. In section 2, we will introduce the principle of the security aspect library, and show its architecture and usage. Section 3 describes an example of application in which security functions have been implemented by invoking aspects in the library. In section 4, we summarize our work, and discuss future efforts relevant to our research.

2 Java Security Aspect Library

Implementing the separation of security aspects from common business logics by AOP only addresses one portion of the problems. It can not implement the reusability of security aspects. With the rapid development of software reusing technologies, Component-Based Software Development (CBSD) becomes more and more popular. Following CBSD, software developers can reuse currently existing components during every phases of software development process to construct a large scale software system. What we are constructing, Java Security Aspect Library

(JSAL), is such a software component at the level of aspects, attempting to provide reusable components in the phase of security implementing.

2.1 Principle

AOP is an approach helpful for software security. AOP-based security framework [4] provides the mechanism for reusing aspects. But there are limited reusable software components for application-level security implemented in AOP. Generic aspect library [5] is a basic container for reusing aspects. In our opinions, the main content of an aspect framework can be implemented as a generic aspect library, such as abstract mechanisms and concrete mechanisms mentioned in [4].

Abstract aspect, with abstract pointcut provided by AspectJ [6], enables us to prepare common security aspects for reusing. A carefully and reasonably organized collection of those aspects would be a reusable and generic security aspect library, which would be a software component for implementing security functions by AOP. For constructing this library, we firstly extract security functions from applications, and then encapsulate these functions into aspects. We also try to apply those rules proposed in [7] to guide the construction of JSAL. When building application, developers can reuse these security aspects through aspect recomposing or overriding.

The security aspect library (JSAL) is simple, flexible and extensible. It can be integrated seamlessly with software development. JSAL is built upon JCE [8] and JAAS [9] in AspectJ. As mentioned above, abstract aspect with abstract pointcut in AspectJ provides the basic mechanism for reusing aspects. Furthermore, popular security packages in Java, upon which JSAL has been built, assure the generality to a certain extent. JSAL has the following features:

- It contains security aspects commonly used by usual applications in Java.
- It supports the reusability and extensibility of aspects.
- It emphasizes the combination of CBSD with AOP.

2.2 Architecture

JSAL is implemented in AspectJ, the most mature Aspect-Oriented Programming implementation as extension to Java. Figure 1 shows the architecture of JSAL. JSAL is composed of reusable security aspects. These aspects reside in the library as abstract aspects in AspectJ. JSAL currently contains the following typical categories of security aspects:

- Encryption/Decryption
- Authentication
- Authorization
- Security audit

Correspondingly, JSAL now comprises four independent parts. Some parts of security aspects can be divided into several subparts by extending. For example, aspects for encryption can be divided according to the types of relevant algorithms, such as DES and AES. There now is a common abstract aspect for encryption/decryption, which defines common operations for encryption/decryption. Concrete aspects in different encrypting algorithms are defined by extending the common abstract aspect.

Security aspects, as shown in Figure 1, comprise codes invoking Java security packages such as JCE and JAAS. These aspects encapsulate complicated security-relevant codes and the process of invoking security packages. For the advantage of AOP, the security aspects enable the applications, which using these security aspects, easy to modify or maintain.

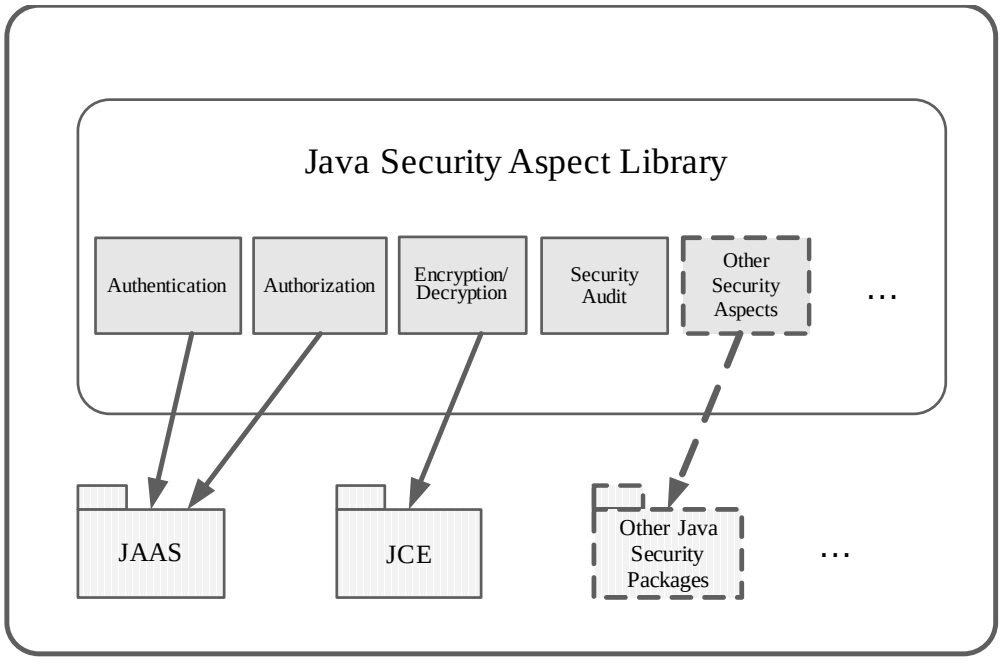

Figure 1: JSAL architecture

2.3 Usage

The following examples will illustrate a security aspect and its usage. The security aspect presented here is an encrypting aspect extended from the corresponding abstract encrypting aspect in JSAL. The abstract aspect for encryption in DES is shown as follows:

```
public abstract aspect AbstractDESAspect {
 public abstract pointcut encryptOperations(String msg);
 public abstract pointcut decryptOperations(String msg);

 public void around(String msg): encryptOperations(msg) {
  //Create encrypter class
  DesCipher enc = new DesCipher();
  enc.savekey("Deskey");
  //Encrypt
  String encryptedMsg = enc.encrypt(msg);
  proceed(encryptedMsg);
 }

 public void around(String msg): decryptOperations(msg) {
  //Create encrypter class
  DesCipher enc2=new DesCipher("Deskey");
  //Decrypt
  String decryptedMsg = enc2.decrypt(msg);
  proceed(decryptedMsg);
 }
```

The abstract encryption aspect defines two abstract pointcuts: encryptOperations and decryptOperations, and adds concrete encryption/decryption advice around both pointcuts. When using this aspect, programmers just need to extend the abstract aspect according to the security policy of concrete application by overriding or defining pointcuts.

Here is an example which will invoke the aspect shown above:

```
public static void main(String[] args) {
   String text = "Hello world!";
// send messages, needs encryption here
   String set = sendMsg(text);
   System.out.println("The send messages are : " + set);
   String dec = recvMsg(set);
// receive messages, needs decryption here
   System.out.println("The send messages are : " + dec);
 }
```

According to the security policy of an application, for example, the string “msg” needs to be protected by encrypting. So we can define the following extended encryption aspect, in which defines the pointcuts that need to enforce encrypting/decrypting operations.

```
public aspect MyDESAspect extends AbstractDESAspect {
  public pointcut encryptOperations(String msg):
    call(String sendMsg(String))
    && args(msg);
  pointcut decryptOperations(String msg):
    call(String recvMsg(String))
    && args(msg);
}
```

The function of those codes shown above is to describe the corresponding security aspect pointcuts. After extending an abstract encryption aspect and defining pointcuts, the compiler provided by AspectJ can weave the aspect with other codes. So the original application can be enhanced with encrypting.

In a word, JSAL provides reusable security aspects for application in the process shown above.

3 JSAL Example

We have simply assessed the usability of JSAL through a practical case upon what Bart De Win, Wouter Joosen and Frank Piessens have done in [10]. They used AOSD techniques to modularize the security features within the jFTPd [11]. Four new aspects were added by them to deal with crosscutting behavior: two aspects (FTPSession, FTPConnectionSecurity) contain the security logic of the FTPConnection class, which includes keeping session state and enforcing the access control policy, and two aspects (FTPHandlerSecurity, FTPHandlerSecurityState) have similar responsibilities with regard to the FTPHandler class.

Based on JSAL, we modified some aspects by extending those abstract security aspects in

JSAL. As shown in figure 2, two original security-relevant aspects (FTPConnectionSecurity, FTPHandlerSecurity) were modified to extend two corresponding abstract aspects (AbstractAuthenticationAspect, AbstractAuthorizationAspect) in JSAL. Furthermore, we also added two new security-related aspects (FTPDataEncryption, FTPAudit). The FTPDataEncryption, which extends the AbstractDESAspect in JSAL, is responsible for encrypting plain data during transmission. And the FTPAudit, which extends the AbstractAuditAspect in JSAL, is responsible for auditing those events relevant to security.

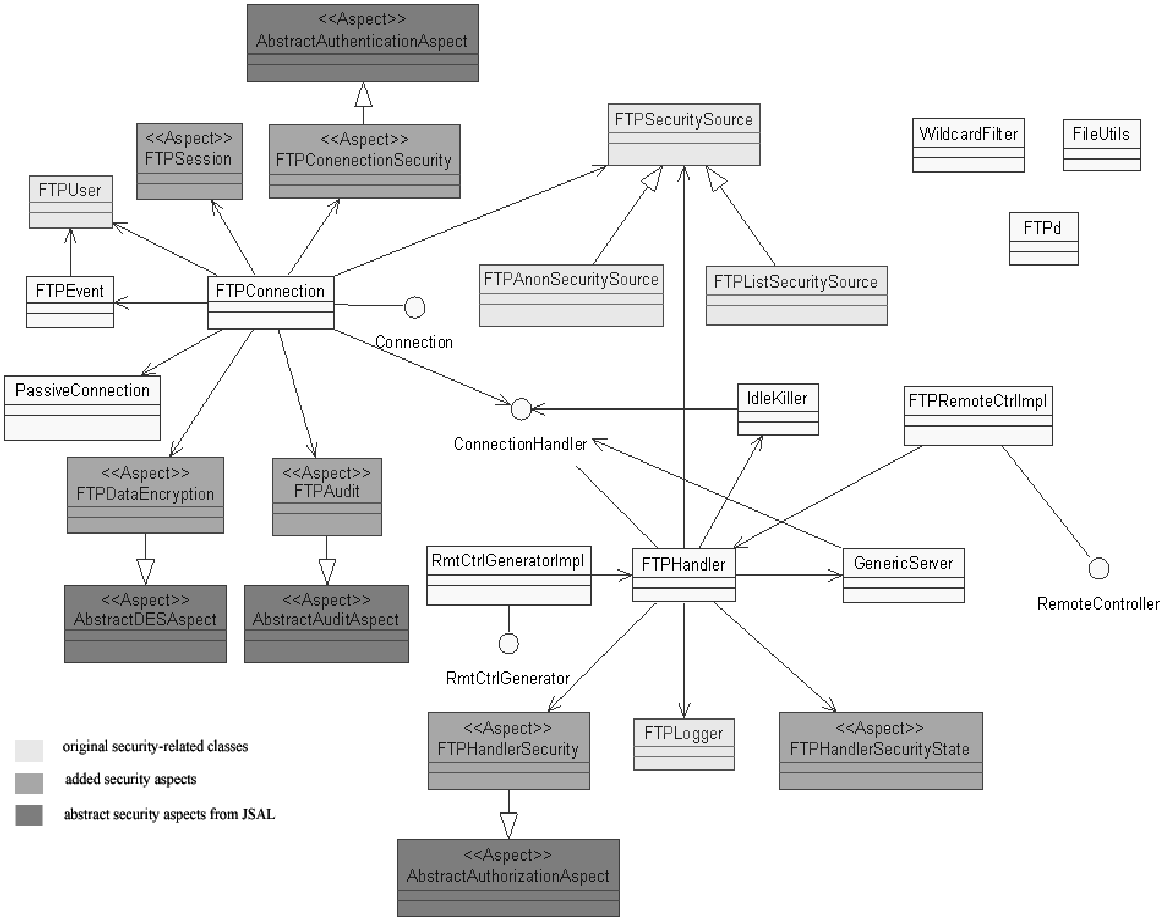

Figure2: jFTPd enhanced with security aspects

Our experiences within this case support what have mentioned of about JSAL. Since JSAL is implemented in AspectJ, so JSAL assures the convenience to achieve a complete separation of the security features with improved modularity. Furthermore, since the concrete security aspects were extended from abstract aspects in JSAL, so JSAL eases the coding of security-related aspects.

4 Discussion and Conclusion

We believe that AOP combined with CBSD is a promising trend for application-level security, such as what is described in [4] and what we have done. Compared with other research efforts on application-level security by AOSD, our efforts are focusing on providing practical reusable

components at the level of aspects in AspectJ. We think that it is a worthy attempt. The main contribution is that JSAL tried to provide reusable security aspects in AspectJ as a practical software component and a prototype of common security-relative API for AOSD. Unfortunately until now we did not do much more for the generality [12] of JSAL, except for choosing popular Java security packages to build upon. For the basis of JSAL, JSAL is generic to some extent, but not enough. In the near future, we will focus on expanding the content of JSAL, and enhancing the generality of JSAL.

While we are pursuing the generality of JSAL, we realize that the generality of security aspects seems to be a hard goal because of the distinct differences between security-related contexts in omnifarious applications. Usually the generality of a software component contradicts its usability and reusability to some extent. So it would seem unpractical to provide some bare aspects in aspect frameworks or libraries just for the sake of generality. Furthermore, those current reusing mechanisms in AOP are similar to inheriting in OOP, so they are not perfectly suitable to those problems which AOP want to solve [6]. Finally, it is another factor which influences on components for AOP that there are few distinct specifications for organizing reusable aspects in AspectJ until now.

Acknowledgments. We are grateful to anonymous referees for their careful comments and suggestions. Also we acknowledge the organizers of AOSDSEC.

References

[1] Bart De Win, Frank Piessens, Wouter Joosen and Tine Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering, Workshop on the Application of Engineering Principles to System Security Design, 2002.

[2] Viren Shah and Frank Hill. Using Aspect-Oriented Programming for Addressing Security Concerns, Supplementary Proc. Of the 13th International Symposium on Software Reliability Engineering (ISSRE'2002), pp.115-119, 2002.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Christina Lopes, Jean-Marc Loingtier, John Irwing. Aspect-Oriented Programming. Proceedings of ECOOP '97, LNCS 1241, Springer-Verlag, pp. 220-242, 1997.

[4] B. Vanhaute, B. De Win, and B. De Decker. Building Frameworks in AspectJ, Workshop on Advanced Separation of Concerns (L. Bergmans, M. Glandrup, J. Brichau and S. Clarke, eds.), 2001, pp. 1-6.

[5] Feng Chen, Policies As Design and Implementation Artifacts For Non Functional Requirements, Master thesis, Department of Systems and Computer Engineering Ottawa-Carleton Institute for Electrical and Computer Engineering, Carleton University September 8, 2002.

[6] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold. An Overview of AspectJ. Appears in ECOOP 2001.

[7] Stefan Hanenberg and Rainer Unland, Using and Reusing Aspects in AspectJ, Workshop on Advanced Separation of Concerns, OOPSLA, 2001.

[8] Sun Microsystems. Java cryptography extension. http://java.sun.com/products/jce/index.html.

[9] K. Koved, A. Nadalin, R. Schemers, C. Lai, L. Song. User authentication and authorization in the java platform. In proceedings of the 15th annual Computer security Applications Conference, December 1999.

[10] B. De Win, W. Joosen and F. Piessens, AOSD & Security: a practical assessment, Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT03), 2003, pp. 1-6.

[11] FTP Server with Remote Administration. http://homepages.wmich.edu/ p1bijjam/cs555 Project/.

[12] B. Vanhaute, and B. De Win, AOP, security and genericity, 1st Belgian AOSD Workshop, Vrije Universiteit Brussel, Brussels, Belgium, November 8, 2001.