Security Role Based Data Encryption for J2EE Web Applications

Heesun Park
SAS Institute, Inc.
Cary, NC 27513, USA

Abstract

This paper addresses end-to-end security for web applications. One aspect of this security is a security-role-based data encryption scheme that enhances the protection of application data from unauthorized access. With the advent of SSL and HTTPS, communication between the web client and server generally is encrypted and the information being exchanged is considered safe during transmission. However, application data on the server side, more often than not, is stored without encryption and is vulnerable to a security breach. J2EE web applications have a security role mapping capability that can restrict access to the web application to specified groups or individuals. The idea behind the application data encryption discussed in this paper is to associate a security role defined for the web application with an encryption algorithm and to use that encryption algorithm when the application stores or retrieves sensitive data. To provide flexibility in assigning an encryption algorithm to a security role, we will propose the use of a Role to Encryption Mapping Table (REMT). We also include a sample routine that encrypts data. This method makes use of the Java Cryptography Extension (JCE) API.

Keywords: Security role mapping, application data encryption, Role to Encryption Mapping Table (REMT), Security Role-Based Encryption Module (SREM)

1. End-to-End Web Application Data Security

Current web applications generally run in a multi-tiered environment. This environment consists of a client tier where the web browser runs, a middle tier that contains an application server where web applications run, and a data tier that holds the data processing servers that control the processing and storage of data. We can look at web application data security in two ways: One way involves looking at the data in transmission. In other words, we need to make sure that information that we put on the wire is properly encrypted so that even if it is sniffed out, it should not be comprehensible. The other way involves looking at application data in storage. Typically, application data is written to the data tier by the web application. With the proliferation of internet and web based applications, there have been significant advancements as far as protecting application data in transmission. But protecting data in storage is still weak at best. In some cases, data in storage does not get encrypted at all and in other cases, data gets encrypted with monolithic encryption server.

The major focus of this paper is to explain how to encrypt application data based on the security role assigned to the user of a J2EE[1] web application. It should be noted that, typically, data encryption from the specific web application is not a preferred way for data in server side storage. So what we are proposing in this paper applies only to the critical web application that handles

sensitive data which warrants extra protection. Since our security role based data encryption gets carried out in the web application itself, it works with any underlying server side data encryption mechanism or device, if any.

Let's briefly examine how we protect application data in transmission while a web application is running. With the addition of the Secure Socket Layer (SSL) protocol [2] to the HTTP protocol [3], the communication between the web client and the web and/or the application server can be encrypted. To a large extent, SSL on HTTP makes the current e-commerce applications possible. The initial SSL handshake uses public key cryptography (PKC) [4], establishes the symmetric encryption algorithm to be used for the remainder of the session, and generates a key for symmetric encryption in a secure way. Note that SSL configuration can be set up between the web client and the web server and between the web server and the application server. In addition, if the application server is configured to make use of a user repository such as an LDAP [5] server, the connection between the application server and the LDAP server can be secured with an SSL flavored LDAP protocol such as secure LDAP (or LDAPS).

As for the protection of application data in storage, you can use a hardware module based implementation or an encryption server based approach. However, both solutions are quite expensive and involved to implement. Also, neither solution is web application specific or associated with the security roles assigned to the web application. What we are proposing in this paper is the security-role-based encryption of the application data so that users who belong to a security role are the only ones who can access the application data.

Figure 1 below shows the end-to-end flow of information in a web application and the potential security vulnerability of the application specific data in storage.

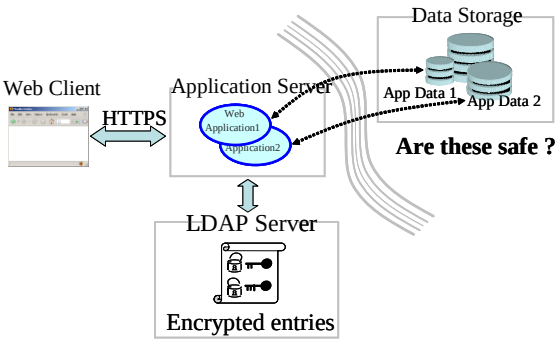


Figure 1: End-to-End Application Data Security

2. Security Role Mapping for Web Applications

The J2EE standard for web applications includes a security provision for web applications. A web application uses a deployment descriptor to indicate to the application server how it wants to issue the authentication challenge, which authentication method to use, and the name of the security role to associate with the application. It is a protection mechanism for the web application itself and is defined through the <security-constraint>, <auth-constraint>, and <security-role> elements in the web application's web.xml (deployment descriptor) file.

Here is an example:

```
<security-constraint>
    <web-resource-collection>
    <web-resource-name>MyWebApp
    </web-resource- name>
     <url-pattern>/*</url-pattern>
     <http-method>GET</http-method>
      <http-method>POST</http-method>
    </web-resource-collection>
  <auth-constraint>
    <role-name>marketingRole</role-name>
   </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>My Realm</realm-name>
</login-config>

<security-role>
   <role-name> marketingRole </role-name>
</security-role>
```

The J2EE security role model consists of two

levels, an application level and a system level. Security role mapping links the two levels together. This is a two-phase process. In the first phase, a web application defines a role name, which is the name of a logical group. The members of that logical group are allowed to access the web application. In our example above, the role name or the logical group name is "marketingRole." The second phase occurs during the deployment process (or after the deployment), when an application level role name is mapped to a set of physical users. These users are found by the application server during a lookup in a user registry or a similar way. The mapping process implementation differs according to the application server, but the logic is the same. Simply put, security role mapping provides fine-grained, dynamic control of access to web applications for the application server administrator.

3. Application Data Encryption from Web Applications

Most J2EE web applications generate application specific data, such as reports in XML form, HTML pages, or raw data for consumption by their own logic. Note that the web application data is the data that web application directly stores in, or retrieves from, the data storage area. One of the reasons for encrypting application data in storage is that if the data storage is compromised or exposed, it is a lot bigger problem than that of someone intercepting pieces of information during transmission.

Even though the Java Development Kit (JDK) provided by J2EE compliant application servers includes the Java Cryptography Extensions (JCE) [6] for encryption and decryption, not many web applications use this capability effectively. It can be used in a general fashion with a fixed encryption algorithm for all web application users, but our goal is to protect sensitive application data with an encryption algorithm that is appropriate for each security role defined in the web application, so that only the members of the security role group are allowed to access the sensitive data. Clearly, there will be multiple web applications in large organizations, and each web application will carry its own security role mapping for its protection. Also, when it comes to the implementation of the encryption logic, we would rather have one common module to handle the encryption and decryption of data, rather than having each web application provide its own version of it. We will call this module that handles encryption and decryption a Security Role-Based Encryption Module or SREM. Obviously, each web application needs to call the SREM with a security role name, the location of the data to be encrypted or decrypted, and possibly a few more encryption property related parameters. Since the SREM needs to support multiple web applications (and multiple security role names), it is necessary to maintain the association between each security role name and the associated encryption algorithm and its properties somewhere. Also, it is desirable that all encryption details be hidden from the web applications themselves. With that in mind, we propose a Role to Encryption Mapping Table (REMT) that maps security role names for a web application to a set of encryption properties, such as name of an encryption algorithm and the data with which encryption keys can be generated. The REMT becomes a key resource of the encryption logic that we are trying to implement, and it should be tightly controlled by the administrator of the web application and IT infrastructure. We will explain the implementation logic of a module that uses this table and the flow of information in the next section.

4. Implementation of an SREM That Uses an REMT

The first step is for an administrator to construct the REMT. In the simplest form, it could a plain text file with security role names and encryption information similar to this:

Table 1: Sample Role to Encryption Mapping Table (REMT) – Plain

Security Role-name	Encryption Algorithm	Encryption Key Data
marketingRole	DES	desKey
salesRole	RC4	Rc4Key
adminRole	DES	desKey

Note that the "Encryption Key Data" column contains the seed value to be used to generate the encryption key in the encryption module; it is not the actual encryption key. In other words, having this by itself does not necessarily mean that you can decipher the encrypted content.

It is possible to use plain text, but using cryptic entries for the encryption algorithms and encryption key data fields is better. This approach enables you to hide the actual encryption algorithm and encryption key data used in encryption and decryption operation. The actual encryption algorithm can be internally resolved in the code itself using the cryptic value obtained from the REMT. Here is a sample REMT with cryptic entries:

Table 2: Sample Role to Encryption Mapping Table (REMT) – Cryptic

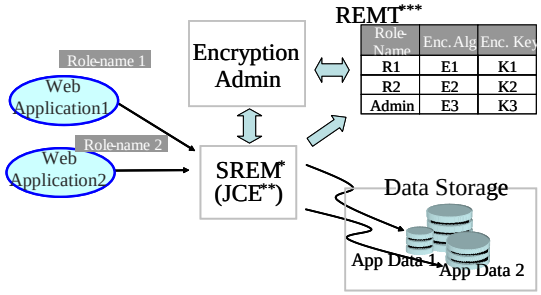
Security Role-name	Encryption Algorithm	Encryption Key Data
marketingRole	E1	K1
salesRole	E2	K2
adminRole	E1	K1

The administrator can maintain the REMT using a separate interface or by an administrator-only function embedded in the SREM itself. Either way, file permissions should set so that the administrator is the only one who has full access to the table, while other users have read access only.

From the web application perspective, the encryption and decryption operations are very much transparent. The application makes a call to the SREM with a security role name and the location of the data to encrypt or decrypt. Then the SREM carries out the task. The SREM reads the REMT and retrieves the encryption information associated with the security role name. If the encryption algorithm and the encryption key data have mnemonic values, then those names can be resolved to meaningful ones inside the SREM. This approach provides extra protection for the stored data even if the REMT becomes exposed.

Note that there is an entry for an "adminRole" in the sample REMT above. This role name should only be used by the administrator to manage encryption and decryption on behalf of the other security role groups. For instance, the administrator might need to create reports for the "marketingRole" group. The administrator sets up the encryption algorithm and the encryption key data in the REMT for those in the "marketingRole" and creates the encrypted reports. Once the reports are created successfully, the administrator can change the REMT entry for himself to something else. Now the encrypted reports are available only to the "marketingRole" group since it has the right encryption information in the REMT.

Figure 2 below depicts the components involved and the flow of information for security role based application data encryption.



*SREM : Security Role based Encryption Module
**JCE : Java Cryptography Extension
***REMT : Role -name Encryption Mapping Table

Figure 2: Role-name based Application Data Encryption

Here is a JCE-based code snippet that gets the encryption information from the REMT and carries out the encryption of application data:

```
// API definitions
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;
import com.ibm.crypto.provider.*;

// Retrieve encryption properties for the role from
// the REMT – determine the encryption algorithm
// and the encryption key to use
…..
rolename = "marketingRole";
encalg = "DES";
// make up the encryption key from "desKey"
enckeydata = "Life_Is_Good";
```

```
….
// Generate encryption key/spec
KeyGenerator
  keygen = KeyGenerator.getInstance(encalg);
byte[] roleKeyData = enckeydata.getBytes();
DESKeySpec
 desKeySpec = new DESKeySpec(roleKeyData);

// set IBMJCE provider
Provider ibmJce = new IBMJCE();
SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance(encalg,ibmJce);
SecretKey rolenameKey =
keyFactory.generateSecret(desKeySpec);

// Create a cipher instance and initialize it

Cipher desCipher;
desCipher = Cipher.getInstance(encalg);
desCipher.init(Cipher.ENCRYPT_MODE,
rolenameKey);


// Prepare for target data and do the encryption
origtext = “The data to be encrypted” ;
byte[] cleartext = origtext.getBytes();
byte[] ciphertext = desCipher.doFinal(cleartext);

// Save the encrypted content to the file system
…
```

5. Management and Encryption of the REMT

In our proposed application data encryption scheme, the REMT plays a key role since without it, application data is not accessible. Therefore, it is important to keep this table in a safe place, and it is imperative to make a backup copy and store it off-line. Also, we need to consider the encryption of this table itself. Even though the encrypted application data cannot be decrypted with this table alone (since the SREM generates the actual encryption key), it is still safer to keep the table in encrypted form.

The administrator is the only one who has full control of the REMT, and it is desirable to keep the encryption of this table independent from the operation of the SREM. One way to do this is to issue a X.509 [7] based certificate to the administrator and have him to use the public key to encrypt the content of the REMT for storage and use the private key to decrypt the table when he makes the table available for SREM operation or makes updates. We talk about certificate based encryption without much qualification here, but for our simple purpose of encrypting the REMT for storage, the use of a self-signed certificate would suffice. We will explore other options as we discuss security-role-based encryption in a Public Key Infrastructure (PKI) [8] environment.

6. Security Role Based Application Data Encryption in PKI Environment

Public Key Infrastructure (PKI) is based on Public Key Cryptography (PKC) [4,9], and its implementation through X.509 certificates. PKC could also be used in Identity Based Encryption (IBE) [10], but it is not commonly used in a PKI implementation.

A certificate can be issued to a user or a computer. It contains, among other things, the subject field that represents the identity of the certificate holder and the public key that can be used in encryption and decryption. Basically, a PKI environment provides a more secure user authentication mechanism as well as the encryption capabilities for the file system and applications. For secure communication on the web, the certificate plays a vital role in the Secure Socket Layer (SSL) handshake and makes HTTP traffic secure. Also, the certificates can be used for authentication for web applications [11].

Security role based encryption of application data can be implemented in a PKI environment by tagging on to its Encrypting File System (EFS) [12] capability. The approach is quite different from the one that we proposed in this paper, but we want to show that we can exploit the EFS’s capability to achieve a similar result. We assume that each web application user gets his or her own user certificate from the issuing Certification Authority of the PKI. In general, the EFS works as follows: EFS encrypts the data with a symmetric encryption algorithm determined internally and then builds the metadata that contains the file handle (or the locator) of the encryption key used in the encryption. The metadata has slots where it keeps the user permission tokens for the encrypted data. A token can be generated by encrypting the file handle with the user’s public key when a user is granted access to data. To access encrypted data, EFS tries

to decrypt the tokens on the permission stack with the user's private key. If the decrypted token matches the file handle of the encryption key, the user can access the encrypted data.

It is feasible to associate the security role mapping of a web application with EFS file permissions. A system administrator can grant permission to the encrypted application data in EFS by employing a user certificate issued in the PKI setup. Handling a large number of files this way could be very cumbersome, though.

It is definitely possible to implement the REMT type of encryption scheme using the certificate issued to represent the security role group. In this case, the table would be called the Role to Certificate Mapping Table (RCMT). The SREM module accesses the RCMT, picks up the certificate, and use the certificate's public and private key to encrypt or decrypt the application data. The certificate can also be used in conjunction with EFS if desired.

7. Impact on Performance

When data encryption gets carried out in the web application layer, it definitely will affect the performance of the application. Impact on performance varies mainly by the size of the data to be encrypted and the choice of encryption algorithm. Other factors include speed of the machine and memory capacity, the memory heap allocated to the application server and the performance of the application server in which the web application is deployed. Our experience based on SSL access (HTTPS) to the web application shows the performance overhead of 5 to 15 percent compared to regular HTTP access. The security role based data encryption by the web application on server side will add more performance burden on server side on top of SSL, but it should be less than that of SSL since it is for server side only. By the way, the data encryption by the web application can be used with or without SSL, but use of SSL is strongly recommended as it provides protection of data in transmission. As noted in the introduction section, this approach is not intended for general server side data encryption, rather, it is intended for the extra protection of the data generated by the sensitive web applications.

8. Conclusion

Internet and web based applications will grow significantly in the future as will the amount of web application generated data. It is apparent that we need more sophisticated web application data protection mechanisms to make sure that application generated data is protected at the same level as our web applications. Also, data in storage needs to be protected by encryption so that it is not usable if stolen or exposed. In this paper, we exploited the J2EE security role mapping feature of web applications and tied it to the encryption of the data produced by the web applications. To make the data encryption process transparent to the web applications, we introduced a Role to Encryption Mapping Table (REMT) that links the security role assigned to the web application to encryption properties that can be used for the actual encryption and decryption of data for storage and retrieval. We have also shown the sharable encryption logic called a Security Role-Based Encryption Module (SREM) that accesses the REMT, picks up the encryption algorithm and the seed value to generate the encryption key, and carries out the encryption and decryption of data for the web application. This process ensures that targeted data can only be accessed by the members of the security group that is mapped to the web application.

Data encryption and protection for web applications can be implemented in other ways including the use of certificates in the PKI environment. The most important thing is to make sure that you provide all the necessary protection for the web application data that you create and manage.

9. References

[1]Java 2 Platform, Enterprise Edition (J2EE) Overview
http://java.sun.com/j2ee/overview.html

[2] SSL protocol version 3.0, March, 1996
http://wp.netscape.com/eng/ssl3/ssl-toc.html

[3] HTTP protocol, June 2007,
http://www.w3.org/Protocols/

[4] "Cryptography and Network Security: Principles and Practice", Second Edition, William Stallings, 1998

[5] Lightweight Directory Access Protocol (LDAP), December 1997, http://www.ietf.org/rfc/rfc2251.txt

[6] JCE : Java Cryptography Extension http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html

[7] X.509 certificate, http://en.wikipedia.org/wiki/X.509

[8] Public Key Infrastructure, PKI, http://en.wikipedia.org/wiki/Public_key_infrastructure

[9] Public Key Cryptography, PKC, http://en.wikipedia.org/wiki/Public-key_cryptography

[10] Identity based encryption from the Weil pairing. by D. Boneh and M. Franklin SIAM J. of Computing, Vol. 32, No. 3, pp. 586-615, 2003, Extended abstract in Crypto 2001, LNCS 2139, pp. 213-229, 2001.

[11] Client certificate and IP address based multi-factor authentication for J2EE web applications, by Heesun Park and Stan Redford, CASCON, 2007, Toronto, Canada

[12] Encrypting File System, EFS, http://en.wikipedia.org/wiki/Encrypting_File_System