Paper 325-2009

SAS® Business Intelligence Web Application Security Configuration Primer

Heesun Park and Brian English, SAS Institute Inc., Cary, NC

ABSTRACT

Securing Web-based resources is one of the biggest challenges for IT today. Almost all IT organizations use security measures through authentication and authorization to protect their Web resources. Thus, it is vital for SAS® Business Intelligence Web applications to integrate within a secure Web environment. This paper explores just that.

SAS Business Intelligence Web applications are implemented based on the SAS® Metadata Server, which typically is tied to the local OS for host authentication, but it can be integrated with an external authentication mechanism already in place for an organization's Web space. The latter is called Web (or trusted) authentication.

Web-based authentication can occur in various Web components such as Web (HTTP) server, reverse proxy security server, or application server. We will examine the pros and cons of various Web security configurations, Single Sign-On (SSO) capability through third-party security packages, and how SAS Business Intelligence Web applications operate within each one.

INTRODUCTION

Security implementation of J2EE Web applications consists of two parts: authentication to control access to the Web application and authorization to control what operation is allowed on resources, such as servers and data, by the authenticated user. This paper mainly focuses on the authentication process for Web applications, which requires coordinating with existing Web infrastructure.

J2EE-based SAS Business Intelligence Web applications use two authentication mechanisms: local OS user-registry-based host authentication, also called SAS authentication, and Web authentication, also called trusted authentication. The most popular user registry for Web authentication is an LDAP directory server, but a flat password file or DBMS is also permitted. To support Web authentication, SAS Business Intelligence Web applications must integrate with an organization's authentication mechanism for the Web space.

The choice of authentication for a Web application is configurable through a pluggable Java Authentication and Authorization Service (JAAS) login module[1]. Web authentication can occur in various Web components such as a Web (HTTP) server, a reverse proxy server, or an application server. The factors in Web security configuration of J2EE Web applications include the authentication mechanism, the location of the authentication challenges, the type of user registry, and possibly Single Sign-On (SSO) capability with a third-party security package or other Web applications.

The first thing that you need to understand is the difference between the traditional Web HTTP server security mechanism and the J2EE Web application security arrangement. HTTP server security was originally designed to protect static documents. On the other hand, J2EE Web applications are fully independent Java programs that run in an application server or servlet container and include a security mechanism or standard called JAAS. Technically, an application server is more than just a servlet container, but the terms are used interchangeably in this paper. J2EE Web applications can function without the involvement of HTTP servers, but, in most cases, HTTP servers and application servers are considered an integral part of Web infrastructure.

The proxy server is another important component in the Web security configuration. Its original purpose was to protect Web users from the Web by restricting access to Web domains and to enhance Web traffic performance by caching resources. A proxy server that protects Web applications and application servers from outside access is called a reverse proxy server. A reverse proxy server that provides security screening through user registry is called a reverse proxy security server (RPSS). The RPSS might be a separate process or a plug-in to the Web server, such as IBM's WebSEAL[2,5] and CA's eTrust SiteMinder Web Server Agent[3].

J2EE Web applications can use different authentication mechanisms and still maintain portability of the code through the configurable JAAS login module. The next section provides an example of JAAS login module usage for a Web application.

In SAS (host) authentication, a Web application might prompt for user credentials and access the SAS Metadata Server for authentication against the local OS registry. For Web-based authentication, you should understand how the user credentials get collected and passed to the Web application. In fact, this is the crucial integration point between the HTTP-protocol-based Web server and the Java-based application server, Java Virtual Machine (JVM), in which a Web application is deployed. Security requirements of the organization and its Web security infrastructure control how to access Web applications.

This paper examines various configuration scenarios, explains important mechanisms, and the pros and cons of each approach.

JAAS LOGIN CONFIGURATION

To understand the authentication scheme and the place of authentication, you should know how JAAS login configuration works in Web applications. JAAS is a Java-based implementation of Pluggable Authentication Module (PAM), and, in its simplest form, it is like JDBC, an abstraction over authentication module providers. The JAAS login modules are the drivers to the Web application security provider. A Web application is configured with different types of JAAS login modules by supplying an authentication option in the property file when the application package gets built. The JAAS login entry, stored in the login.config file, is part of the service configuration file for the Web application, but is not a part of the portable war file.

Suppose your Web application supports two JAAS login modules, host authentication and Web authentication, and the Web application is based on its own metadata server that supports both authentication mechanisms. The following example uses a SAS 9.1.3 implementation of JAAS login configuration in the Web application properties file to build the JAAS login:

```
#Host authentication through local OS
 $LOGON_DOMAIN$=DefaultAuth
$AUTH_MECHANISM$=host

#Trusted authentication through Web components
 $LOGON_DOMAIN$=web
$AUTH_MECHANISM$=trusted
```

The following two JAAS login entries use the alias name of PFS:

```
#JAAS login host authentication through local OS
PFS {
     com.sas.services.security.OMILoginModule
    "host"="myhost.mynode.com"
    "port"="8561"
    "repository"="Foundation"
    "domain"="DefaultAuth";
 }

#JAAS login trusted Web authentication
PFS {
    com.sas.services.security.login.TrustedLoginModule
    "host"="myhost.mynode.com"
    "port"="8561"
    "repository"="Foundation"
    "domain"="web";
    "trusteduser"="trustme"
    "trustedpw"="{abc001}U0FTcHcx"
};
```

The Web application creates LoginContext based on the JAAS login module and then submits the login request to the SAS Metadata Server. The login configuration for trusted authentication uses `trusteduser` and `trustpw` for an initial connection to the SAS Metadata Server because the user credentials for a valid user are unknown. For trusted authentication, only the remote user name is available to the Web applications.

The JAAS login mechanism is stackable and is used by the application server. The implementation of JAAS login configuration differs by the application server. WebSphere uses system login and application login, while WebLogic uses authentication provider. You can also use a JVM argument.

For SAS 9.1.3, the application login (alias name PFS) uses the authenticated user name that is passed in through the RemoteUser field of the HTTP request. SAS 9.2 uses a more secure process. The SAS JAAS login module is integrated into the application server's system login stack. The JAAS Subject object, created by the application server to process and handle authenticated users as the Principals object in the Subject, is also used. However, the setup is more difficult, and is beyond the scope of this paper.

This paper now focuses on trusted Web authentication mechanism in terms of Web components and Web application interaction.

AUTHENTICATION THROUGH APPLICATION SERVER

The J2EE 1.3[4] standard has a provision for Web applications to instruct the servlet container on how to issue the authentication challenge, the authentication method to use, and the name of the security role to associate with the application. The instructions are defined in the application's deployment descriptor file, web.xml.

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

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

The J2EE security role model consists of two levels: application level and system level. The security role mapping connects the two levels in a two-phase process. First, a Web application defines role name, `webuserRole` in the example, which is a logical grouping of users. Next, the actual security mapping occurs during the deployment process where application level roles (and realms) are mapped to the system level roles (and realms) that are defined on the application server. Simply put, security role mapping provides the dynamic mapping of users for Web application access during or after the deployment of Web application.

The <login-config> element defines the authentication method to use. The BASIC auth-method causes the application server to issue authentication challenge in a default pop-up dialog box that prompts the user for a user name and password. The user information is then placed on the HTTP request authorization header.

The following example includes a user-friendly FORM custom login page in place of the default pop-up dialog box:

```
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>My Realm </realm-name>
    <form-login-config>
      <form-login-page>/login.html</form-login-page>
      <form-error-page>/error.jsp</form-error-page>
    </form-login-config>
  </login-config>
```

The login page uses the pre-defined variable names for the action servlet, user name, and password to properly communicate with the application server. They are j_security_check, j_username, and j_password, respectively.

This mechanism requires that the application server connect to the user registry to validate user credentials. The user registry might be the local OS system, a Lightweight Directory Access Protocol (LDAP) server, DBMS, or a Shared File System. The LDAP server is the most popular user information store. Once authenticated, the servlet container places the user name in the request object. A Web application makes a request.getRemoteUser() call to get the authenticated user name. Then, the application server can optionally invoke the JAAS login module to validate the Web authenticated user with its metadata server for authorization of resources later.

Because Web applications are protected by themselves and the servlet container, the HTTP server might have to use its own authentication service to protect static documents or other resources that are under its control.

Figure 1 depicts the flow of control where the authentication challenge is initiated by the Web application, issued by the application server, and authorized through its metadata server.

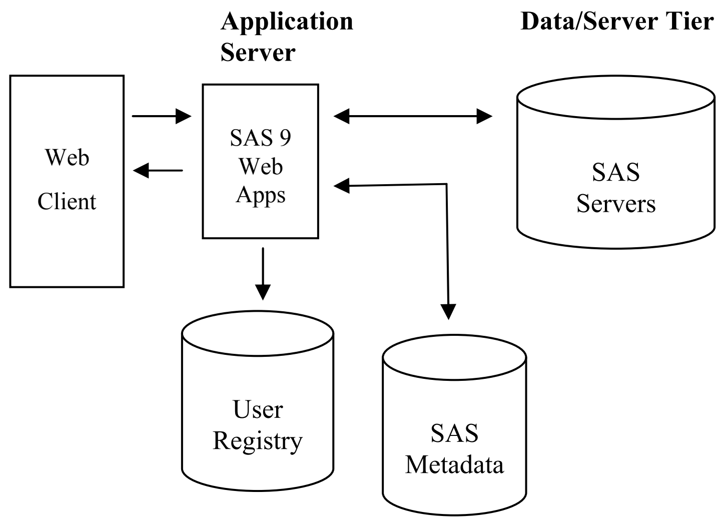

Figure 1: Authentication through an Application Server

AUTHENTICATION THROUGH HTTP SERVER

Some organizations want to control user access to their static documents, as well as dynamic resources such as J2EE Web applications, through their HTTP (Web) server. In this case, the HTTP server is configured to force the browser to issue the authentication challenge for protected resources by adding directives in the httpd.conf file:

```
<Location /mywebapp/ >
  AuthName "My_realm"
  AuthType Basic
  AuthUserFile “C:\Apache\realm\password”
  require valid-user
</Location>
```

The HTTP server supplies the tool to create a simple password file where the user name and password are stored. The Apache Tomcat tool is called htpasswd. In the example, the Web application, mywebapp, uses basic authentication and the password file C:\Apache\realm\password to validate the user name and its password.

Usually, the servlet container provides the plug-in module that maintains communication between the HTTP server and the servlet container. This module is also responsible for the request object surfacing the remote user name (from the Web application perspective) in the application server’s private data structure. When user authentication is based on user information maintained by the HTTP server (a simple password file), Web applications lose the capability of user security role mapping provided by the application server. One advantage of this approach is that the HTTP server’s authentication service controls access to static and dynamic Web resources. The potential drawback is the lack of authorization capability. For example, the LDAP user registry is not fully supported because the HTTP server is designed to serve static documents that usually do not require any rigorous authorization. When the Web application relies on a comprehensive authorization service through its own metadata server, user authentication through the HTTP server is adequate.

Figure 2 depicts the flow of control where the authentication challenge is initiated by the HTTP server and authorization is handled through its own metadata server.

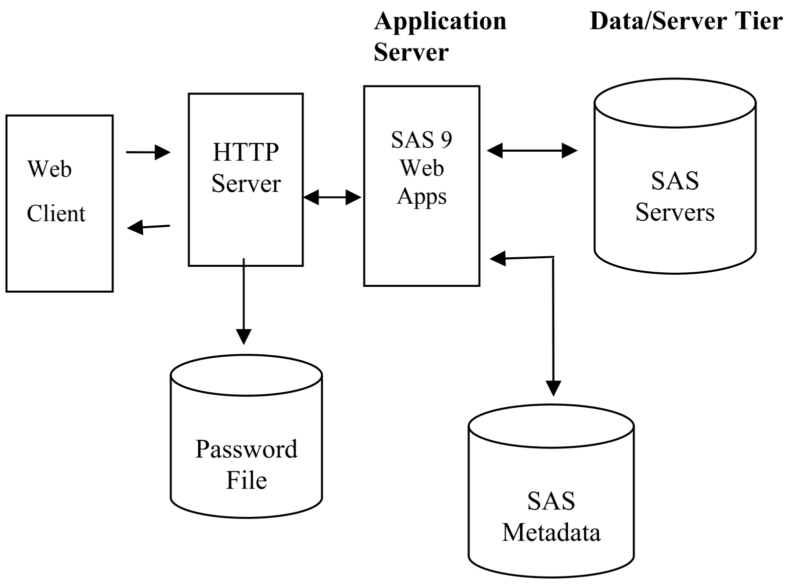

Figure 2: Authentication through HTTP Server

AUTHENTICATION THROUGH REVERSE PROXY SECURITY SERVER (RPSS)

Authentication using the RPSS protects Web resources from external access through an array of authentication methods that include basic, forms, certificates, and tokens. By providing authentication services for different types of Web resources and Web applications, the RPSS easily supports the SSO (Single Sign-On) capability for participating applications. Static resources are protected through its connection (called a junction in WebSEAL) to the HTTP servers. Dynamic Web application resources are protected through its connection to the application servers. Typically, a sophisticated user registry is used, such as LDAP server, for user authentication and resource access authorization.

The Web application uses trusted Web authentication through the RPSS. A tricky part is how user credentials are handled between the RPSS and the application server. The application server relies on the RPSS for user authentication. Therefore, an entity must be set up in the application server to receive the request from the RPSS and place the user information into the application server's data structure. For a WebSphere application server, this entity is called Trust Association Interceptor (TAI). For a WebLogic application server, this entity is called Identity Asserter (IA)[6]. In both cases, user credentials are encrypted and the password is omitted.

Figure 3 depicts the flow of control where the authentication challenge is initiated by the RPSS and the authorization is handled through its own metadata server.

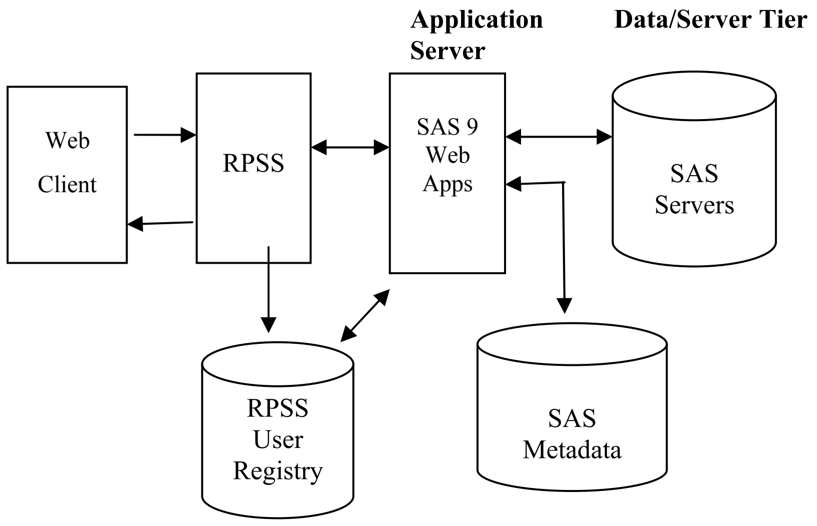

Figure 3: Authentication through Reverse Proxy Security Server (RPSS)

DOUBLE AUTHENTICATION

So far, user authentication has occurred through a third-party security provider and the use of authenticated user credentials in the Web applications. If an organization agrees, then implementing two levels of authentication is permissible. In this case, a Web application might use the SAS Metadata Server to authenticate users against the local OS. The first level (or system level) authentication occurs through the RPSS or HTTP server that has its own user registry. You could consider this a system level security checkpoint to determine whether the user is valid to enter the Web domain. For Web applications, second level (or application level) authentication is provided by the Web application and the application server. This provides an extra layer of security and more flexibility in security role mapping. Double authentication is a good practice when the Web application handles sensitive information.

SSO (SINGLE SIGN-ON) SCENARIO

An SSO scenario uses externally authenticated user identities for multiple Web applications. After a user is authenticated, a client (a browser session) can access multiple Web applications that share the same authentication method and user registry without getting a separate authentication challenge (login prompt).

SSO might be achieved by using a special token that is created and supported by the RPSS or the application server. After the authenticated user reaches the application server, the token is created in conjunction with the interceptor (or application server agent). Then, the token is passed back to the client, typically as a cookie. For subsequent requests to Web applications, the RPSS checks the token first. If the valid token exists for the session, then the authentication challenge is bypassed. Examples are WebSphere and WebSEAL's Lightweight Third Party Authentication (LTPA) token and SiteMinder's SMSESSION cookie.

You can also achieve SSO by using a secret (or trusted) user name and password between the RPSS and the application server. After the RPSS authenticates the user, the user information is kept and no additional login challenges are issued. In turn, the trusted user name is used to connect to the application server. On the receiving end, the application server validates the trusted user credentials from the RPSS. Once the user is validated, the user credentials are accepted from the RPSS. Then, the application server uses the credentials to prepare its private data structure for consumption by Web applications.

CONCLUSION

Web application security is primarily controlled by the data that the Web application is based on and the way the data is presented. You can set up very sophisticated role-based security mapping for the Web applications or minimum security similar to a very simple static Web page. This paper explained where and how authentication for a Web application occurs. You were shown how the user credentials that are collected in the HTTP protocol end up in the

Java object for Web applications. The pros and cons of various configurations were discussed. You can use this as a reference point to determine the organization's Web application security framework. For the organizations that have established Web infrastructures, you have seen how the Web applications might fit into the existing security framework.

Many Web applications are based on a metadata server that provides choices of authentication service and a specific resource authorization service with a user-friendly interface. Host authentication through its metadata server provides a simple, very powerful independent Web application. However, use of trusted authentication can seamlessly integrate with existing infrastructure and with other Web applications.

REFERENCES

[1] Java SE Security:
http://java.sun.com/products/jaas/

[2] IBM WebSEAL 5.1:
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame2.doc_5.1/am51_webseal_guide10.htm

[3] CA eTrust *SiteMinder Web Agent Installation Guide, v6.0* (2004) ships with SiteMinder

[4] Java EE Specification:
http://java.sun.com/j2ee/index.jsp

[5] IBM WebSphere V5.0 Security, WebSphere Handbook Series:
http://www.redbooks.ibm.com/redbooks/pdfs/sg246573.pdf

[6] WebLogic Identity Assertion Provider
http://edocs.bea.com/wls/docs81/ConsoleHelp/security_defaultidentityasserter_general.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Heesun Park
SAS Institute Inc.
e-mail: sashsp@sas.com