Security and SilverStream eXtend

November 19, 2001

Overview

The SilverStream eXtend™ product line is a set of J2EE applications and tools that enable the creation of other J2EE applications, particularly those architected around Web Services. Such applications are inherently distributed and usually of enterprise scale, so security is typically very important to those people creating and deploying the applications. This paper will explore the following common security topics as they relate to J2EE, Web Services, and SilverStream eXtend:

- Authentication
- Access Control
- Data Confidentiality and Integrity
- Other Security Topics
    - Non-repudiation and Auditing
    - Protection from Viruses
    - Protection from Denial-of-service Attacks
    - Single Signon
    - Firewalls
    - Maintenance

We will cover the J2EE and Web Services specifications and the particular implementation choices and extensions defined in all eXtend products. We will describe what eXtend provides and which features you can replace or augment. We assume basic knowledge of J2EE and Web Services technologies but not expertise.

Authentication

Authentication is the process of verifying the identity of an entity, which can be a user or another program. The entity, once identified, is commonly referred to as a **principal**. In J2EE, authentication happens in a few ways:

- Web users communicating with a Web container from a browser will use standard HTTP mechanisms to authenticate. The servlet specification describes three methods to be supported.
- J2EE client applications authenticate using mechanisms built into the client container and make use of some JAAS technologies.
- EJB containers authenticate callers whether they are from client containers, Web containers, or other EJB containers.
- Resources authenticate to back-end systems such as databases or Enterprise Information Systems typically by passing a username and password when creating a connection.

Web Container Authentication

The servlet specification describes the following authentication mechanisms that Web containers support:

- HTTP Basic authentication
- HTTP Digest authentication
- Form-based authentication
- HTTPS Client authentication

HTTP Basic authentication is what is commonly used on the Web. A client makes a request of the server (Web container). If the request requires authentication, the server responds with an HTTP 401 response code. The browser receives the response, prompts the user for a username and password, then resends the request to the server with an added header called **Authorization**. The header includes the entered username and password, separated by a colon and base64-encoded. Despite the name base64 encoding, this should not be confused with encryption; it's not. This method is completely susceptible to eavesdroppers sniffing on the wire and is not secure. If SSL is used to encrypt the traffic, then it is reasonably secure — depending on the level of the encryption used. Regardless of its flaws, HTTP Basic authentication is by far the most commonly used authentication mechanism on the Web. Web containers are required to support it, and all browsers support it.

HTTP Digest authentication is similar but instead of sending the username and password base64-encoded, a hash of them is sent. This is only slightly more secure. Eavesdroppers cannot get the actual password, but do learn a string that can be used to replay a message, which allows them to authenticate themselves, as you, without your knowledge. Digest is not commonly implemented in browsers; Web containers are not required to support it.

HTTP Basic and Digest authentication are both defined in RFC 2617.

Form-based authentication is very similar to the above but allows an application to create a customized login screen instead of using the standard browser login dialog. While this capability may not seem that important, it is required of all Web containers. Instead of returning a 401 status code when login is required, the container sends back a status code for an HTTP redirect to a login page. This page contains an HTML form with fields named `j_username` and `j_password` and with action set to `j_security_check`. The browser displays the form and the user fills it out and posts it back to the container. The container recognizes the well-known form fields and performs the login.

Form-based security is roughly the same as HTTP Basic since the username and password are sent over the wire in clear text, so you should use SSL for anything secure. When building the application, the URL of the login page is specified in the deployment descriptor of the WAR file. Here is an example showing how the form should be coded:

```
<form method="POST" action="j_security_check">
   <input type="text" name="j_username">
   <input type="password" name="j_password">
</form>
```

HTTPS Client authentication uses SSL (Secure Sockets Layer) and public key certificates to authenticate a client user. This can result in strong encryption and can be quite secure. Netscape invented SSL to help secure the Web. SSL Version 2.0 was the first commonly implemented version. Version 3.0 is commonly used today. In addition, the IETF defined TLS (Transport Layer Security) in RFC 2246 as a successor to SSL. TLS is now available in the latest versions of most browsers.

SSL and TLS work at the socket layer of the application, establishing a secure TCP connection. This means authenticating the server, possibly encrypting the communications channel, and optionally authenticating the client. Authentication is performed using public key encryption (which will not be described here). Both SSL and TLS allow for a negotiation of encryption mechanisms used by the client and server; these mechanisms are referred to as **cipher suites**. Some cipher suites define weak encryption and therefore offer less security; others are stronger. All J2EE-compliant Web containers must support HTTPS Client authentication and the following relatively weak cipher suites:

- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

Client Container Authentication

The J2EE specification defines only broad authentication requirements for a client container to support. A J2EE client application must be able to authenticate to Web and EJB containers. The container must “provide an appropriate user interface” to gather authentication information from the user. In addition, if an application specifies (in the deployment descriptor) the name of a class implementing the JAAS interface `javax.security.auth.callback.CallbackHandler`, then the container must call this class when trying to authenticate the user. This provides the application with a means to implement whatever authentication mechanism it requires.

EJB Client Authentication

The EJB 1.1 specification set requirements for authenticating EJB clients, but did not prescribe how it was to be done. EJB 2.0 (required in J2EE 1.3) specifies the use of CSIv2 (Common Secure Interoperability Version 2, OMG’s security specification for CORBA) to transfer authentication information from client to server in a standard way, allowing vender interoperability. This mechanism is similar to HTTP Basic authentication and was designed with the idea of using an SSL-like mechanism for securing the communications channel. At deployment time, EJBs can be configured to require client authentication. In addition, EJBs can be configured at deployment time to run as a specified principal, not merely as the principal of the caller. The EJB 2.0 specification also requires that containers support a means to establish a transitive trust relationship between one or more Web and EJB containers.

Resource Authentication

Resources in J2EE refer to external systems such as databases, Enterprise Information Systems, and JMS systems. They are not precisely defined, but think of them as things you access by getting connections via JDBC, Connectors, and JMS. For each of these you can choose container-managed or component-managed authentication. The choice is specified in the application component’s deployment descriptor.

- In **component-managed authentication**, the application code must call the appropriate (possibly proprietary) APIs to do the authentication.
- In **container-managed authentication**, it is the container's responsibility to do the authentication, usually with deployment information.

J2EE requires that containers support both APIs for specifying the authentication information at runtime and a means to specify the authentication information solely using deployment information.

Web Services Authentication

The Web Services technologies do not define much in this area aside from depending on standard HTTP mechanisms including HTTPS support, which is already included in J2EE.

SilverStream eXtend Support

eXtend Application Server

eXtend Application Server implements HTTP Basic, Form-based, and HTTPS Client authentication. It can be configured to require authentication on every HTTP request. It includes a pure Java implementation of SSL 3.0 and TLS 1.0 and supports both RSA and DSA certificates. Requirements for client certificates can be configured in several ways:

- Certificate not requested or required
- Certificate requested but not required
- Certificate requested, not required; auto-add if not in database
- Certificate required, auto-add if not in database
- Certificate required; anonymous if not in database
- Certificate required for known user
- Certificate required for known user, but user remains anonymous

While user registry mechanisms are beyond the scope of J2EE, an application server must have some means of managing the accounts of registered users. eXtend Application Server contains a user registry that stores users in the SilverMaster database. Alternatively, the server can be configured to use the registry from one or more of the following security systems: Window NT, LDAP, and NIS+. The SilverStream Management Console (SMC) is used to administer the SilverStream user registry and can also be used to configure server certificates.

Version 4 of eXtend Application Server will include a jBroker ORB that includes a complete implementation of CSIv2. The application server's EJB container will support the requirements for propagation of the caller’s identity and the definition of a run-as principal at deployment time. It also will support the configuration of transitive trust relationships.

eXtend Application Server includes all required support for JDBC and (in Version 4) Connectors. Usernames and passwords can be specified either programmatically or at deployment time by specifying authentication information in the deployment plan.

The application server ships with the jBroker MQ implementation of a JMS server, which supports the standard JMS APIs for connection creation, including passing a username and password. The JMS server includes its own user and group registry. The `JMQSecurityAdmin` interface allows the management of users and groups. Version 2.1 of jBroker MQ will allow for integration with additional authentication systems, such as LDAP.

eXtend Composer

eXtend Composer consists of a design-time GUI designer and a runtime system to access back-end resources using a connection pool mechanism. The runtime system is a J2EE application capable of being deployed to a variety of application servers. Both the design and runtime systems provide authentication facilities. Typically, all connections share a single user identity whose credentials are defined at deployment time. Both username/password and certificate-based authentication are supported. In addition, the GUI designer is capable of designing components that use credentials obtained out-of-band when establishing connections. For greatest flexibility, developers can override or re-implement the `IGnvConnectionFactory` interface and establish connections however they want.

eXtend Director

eXtend Director consists of design-time GUI tools and several runtime subsystems. The runtime subsystems are also J2EE applications capable of being deployed to a variety of application servers. The runtime subsystems depend on the Web and EJB containers to provide authentication. eXtend Director uses a user registry implementation that allows different "realms" to be plugged in by implementing the `EbiRealm` interface. Implementations exist to connect to the eXtend Application Server user registry, to WebLogic realms, and to WebSphere's equivalent. Developers are free to extend any of these implementations or to provide their own.

The Portal subsystem’s Wireless Transcoding Engine allows communication with clients that are Web browsers as well as other wireless devices. In the WAP (Wireless Access Protocol) world, a separate gateway is responsible for converting WAP to HTTP. Such gateways typically support features needed to enable the use of HTTP Basic and Form-based authentication. WTLS (Wireless Transport Layer Security) is the equivalent of SSL in the wireless world and provides for the use of x.509 certificates that are usable in HTTP Client authentication.

Access Control

Access control (also known as authorization) is the next issue after authentication. After you know who someone is, you can determine what he or she is allowed to do or, more likely, if they are allowed to do what they just requested.

J2EE addresses access control by using **security roles**. Permission is granted to perform an action if the user principal is in the role allowed to perform the action. Roles are listed in deployment descriptors of J2EE archives, and at deployment time these roles are mapped to real accounts or groups in the deployed environment.

Roles are mapped to actions in one of two ways, either declaratively in the deployment descriptor or programmatically by making API calls. The methods `EJBContext.isCallerInRole(role)` and `HttpServletRequest.isUserInRole(role)` return true if the calling principal is in the specified security role.

Note that both of these schemes bind a role permission requirement to a method. J2EE refers to this as class-based access control. There is currently no facility for what J2EE calls instance-based access control, which is a mechanism to restrict access based on specific data, not on a method. For example, a class-based access control mechanism would allow Alice to look up the account balance of any account by allowing Alice to run the method `getBalance()`. An instance-based mechanism would allow Bob to look up his own account's balance but not anyone else's, by giving him access to his account object.

However, J2EE does provide some tools for developers to implement such a scheme themselves. The two methods `HttpServletRequest.getUserPrincipal()` and `EJBContext.getCallerPrincipal()`return the principal of the calling user, allowing the application to perform some authorization check based on the user's identity. Note that in the case of an anonymous user, these methods behave differently. `HttpServletRequest.getUserPrincipal()`always returns null if the user is anonymous, but `EJBContext.getCallerPrincipal()` is required to always return a valid principal (never null). An EJB container is free to implement whatever scheme it wants to represent anonymous users.

The Web Services specifications do not address access control at all. Currently Web Services need to piggyback on the mechanisms of the underlying technologies used for the implementation, such as J2EE.

SilverStream eXtend Support

eXtend Application Server

eXtend Application Server implements all the required declarative and programmatic access control mechanisms. The EJB container identifies an anonymous user with a single distinguished principal named *anonymous*.

JMS ACLs (Access Control Lists) have the following three permissions:

- Consume –- Permission to receive messages or browse a queue, and to subscribe to topic messages
- Produce –- Permission to send messages to a queue, and to publish messages to a topic
- Manage –- Permission to manage jBroker MQ; in the current release, only users belonging to the administrator group have the Manage permission

The ACLs can be changed using a GUI console as well as using standard `java.security` APIs.

eXtend Director

eXtend Director uses ACLs to define authorization at several different levels. The permission set (for example, Read, Write, Create, Update, and Delete) is dependent on the specific ACL. It is stored as meta-data for the ACL, and is extensible using the `EbiSecurityMetaDelegate` interface. All ACLs are stored in the database and can be accessed using the `EbiSecurityAclDelegate` interface.

There are three levels of ACLs:

- A Locksmith ACL with one permission, Protect. This ACL applies to all elements in all Director subsystems. This permission is the ability to change ACLs.
- Individual ACLs for administration of each subsystem, typically with Protect, Read, Create, Update and Delete permissions.
- Element Type ACLs at the subsystem level. Currently, only the Content Management subsystem defines Element Type ACLs and does so for folders, documents, and repositories.

All ACLs can be changed using the `EbiSecurityAclDelegate` API. In addition, Director includes the HTML-based PAC (Portal Administration Console), which can be used to edit the Locksmith and subsystem administration ACLs. The Content Management Element Type ACLs can be edited using the HTML-based PMC (Portal Management Console), which is used to manage the Content Management subsystem.

The Portal subsystem uses a built-in role-based authorization mechanism for pages, components, and workflow processes. Roles and their mappings to users or groups are defined in XML files contained in ResourceSets. For example:

```
<security-role>
    <display-name>System Administrator</display-name>
    <description>Portal App Admins1</description>
    <user-map>
           <principal>administrator</principal>
           <principal>jsmith</principal>
    </user-map>
    <group-map>
           <principal>administrators</principal>
    </group-map>
</security-role>
```

A page, component, or workflow process can (through their descriptor) require that the user be in a specified set of roles. For example, here's a descriptor for a portal page that requires a user to be in either the manager or the administrator role to run the Hello World page:

```
<portal-page>
    <display-name>Hello World</display-name>
    <description>Hello World Page</description>
    <categories>
           <category>UserPages</category>
    </categories>
    <flush-immediately>false</flush-immediately>
    <mime-type>text/html</mime-type>
    <style-name>HelloWorld</style-name>
    <file-name>HelloWorld.xml</file-name>
    <run-role-map>
           <role-name>manager</role-name>
           <role-name>administrator</role-name>
    </run-role-map>
</portal-page>
```

All runtime ACL-based or role-based authorization checks are performed by calling the `EbiSecurityDelegate` interface. Developers can call these checks in their own code as well.

eXtend Composer

eXtend Composer generates components that can be deployed as servlets, EJBs, or Web Service-based services. These services have no additional access control requirements or capabilities beyond what J2EE defines for them.

Data Confidentiality and Integrity

Data confidentiality is concerned with keeping others from seeing sensitive data. You might confuse it with access controls but it is usually different. It usually involves encrypting data traveling over a wire, preventing eavesdropping while allowing authenticated users to perform authorized actions. It could also be encrypting data in persistent storage so that attackers, even if they get in, still cannot read the data.

Data integrity is similar: it involves ensuring that data is not corrupted or modified while the data is in transit. Encrypting the data is sufficient to prevent it from being altered; if it is, it will not decrypt properly. Some systems that require integrity but not confidentiality will encrypt a checksum of the data, which is much faster than encrypting all the data.

J2EE covers data confidentiality by requiring that Web containers support HTTPS (SSL). In addition, the EJB 2.0 specification requires the support of the following cipher suites to be used with IIOP over SSL 3.0 or TLS 1.0:

- TLS_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

According to the EJB 2.0 specification, both EJB clients and servers can be configured to require a secure transport, providing confidentiality, integrity, or both. At runtime, the client and server perform a negotiation to determine if a secure transport is required.

SilverStream eXtend Support

eXtend Application Server

eXtend Application Server supports the following SSL cipher suites:

- SSL_NULL_WITH_NULL_NULL
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES_40_CBC_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA

- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_EXPORT_WITH_DES_40_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

At the present time, jBroker MQ does not support the use of SSL for its communications.

eXtend Composer

The eXtend Composer Designer can connect to running servers, and, if SSL is required, supports the following cipher suites:

- SSLParams.SSL_RSA_WITH_RC4_128_MD5
- SSLParams.SSL_RSA_WITH_RC4_128_SHA
- SSLParams.SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSLParams.SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSLParams.SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

eXtend Director

eXtend Director uses facilities of the application server it is deployed on to provide data confidentiality and data integrity. When using the Transcoding Engine to communicate with WAP devices, WTLS is typically used to provide confidentiality from the end-user client.

Other Security Topics

Non-repudiation and Auditing

Currently, there are no J2EE or Web Services requirements for support of non-repudiation or auditing systems. Non-repudiation is the property that once you have authenticated something you cannot subsequently refute that you did. For example, if Alice possesses a digitally signed contract whose signature verifies with Bob's public key, then Bob can't claim he didn't sign it since no one else is able to use his private key to do that. Auditing systems are logging systems, but with the added property that the logs are not alterable. The idea is that if an attacker does manage to break in and do something, if the action is logged you don't want him to have the ability to alter the logs to cover his actions.

The eXtend platform currently offers no built-in support for these features. The application server does provide logging facilities, but these logs are not tamper-resistant, as a true auditing system would require.

Protection from Viruses

Currently, there are no known viruses that can infect the eXtend platform. We believe this is not coincidence. The eXtend platform is written in Java, which is a typesafe programming language with bounds checking. There is no possibility of the buffer overflow problems common to C, C++, and other languages that most viruses exploit.

eXtend Application Server comes with its own built-in Web server, so you need not run other virus-susceptible Web servers for your application. We have seen several sites that have been attacked by various viruses (such as NIMDA or Code Red), and the server has performed well. In the worst case the server reported a benign error message about receiving malformed HTTP requests.

Protection from Denial-of-service Attacks

There is a vulnerability to denial-of-service attacks from too many requests, but this is the case with all servers running on a public network. For an overview of denial-of-service attacks and a list of some network-level precautionary measures you can do to reduce your exposure to them, see:
http://www.cert.org/tech_tips/denial_of_service.html.

Single Signon

A commonly requested capability is that of single signon. Unfortunately, it is not a very well-defined request. Single signon is the desire to be able to access multiple applications and resources while forcing the user to login only once. This is easy if all the systems are built together and well-integrated, but in most cases multiple systems come from different venders, use different technologies, and have been implemented at different times. As such, they use different security mechanisms, particularly for authentication.

In one sense, eXtend Application Server can do single signon. Because multiple application servers (not configured in a cluster) can all be configured to use an LDAP (or other external) user registry, by using HTTP Basic authentication to one server you'll send your credentials to each, requiring only a single login. Of course, this doesn't take into account an application that needs to pass those credentials to other back-end systems.

With the popularity of applications that use a Web browser as a front-end, a class of single signon solutions that leverage HTTP authentication has become popular. For example, getAccess by Entrust is a product that, among other things, plugs into a Web server and intercepts authentication requests, then uses its own authentication service to validate the user. getAccess also intercepts other requests and performs authorization checks by matching on the requested URL. This allows you to secure an application (regardless of its back-end) by performing access control on the URLs it exposes. eXtend Application Server contains Web Server Integration (WSI) modules that plug in to IIS, Netscape, and Apache servers. Using WSI modules allows a Web server to proxy for the application server and allows products that integrate with Web servers, such as getAccess, to integrate with the application server.

Firewalls

Firewalls are commonly used to restrict remote access to a machine. It is a broad term that can describe many different functions. Some firewalls restrict clients to be from specific addresses; some restrict access to specific ports. Certain protocols work by using well-known ports, and if you close that port you are not susceptible to requests or attacks on those protocols. eXtend Application Server separates its runtime operation from its design and administration functions. Each of the three can be configured to run on different ports, allowing a firewall to be configured to deny access to these capabilities. For example, the server can be configured so that administration functions can be performed only from within the corporate network.

Maintenance

Finally, the most common security issue is users not properly configuring their systems. Security is often an afterthought or is neglected altogether. It is important that system administrators take all the recommended steps to secure their systems and keep up-to-date with patches as they become available. With some products, patches are released frequently so this can be quite a burden, but so far eXtend Application Server has not had to release a security patch.

By default, Version 3.7 of the application server installs the server totally secured, rather than relying on administrators to lock down every little thing. Still, the best thing any organization that cares about security can do is to have a security plan and a regular review to make sure it is being followed. The SilverStream eXtend Application Server *Administrator's Guide* provides a security checklist to help you secure your environment.

For More Information

Securing Your Web Applications
SilverStream eXtend Application Server Administrator's Guide

SilverStream eXtend product information
http://www.silverstream.com/extend

Strategic partner information
http://www.silverstream.com/partner

SilverStream Developer Center
http://devcenter.silverstream.com
Visit the SilverStream Developer Center for additional information on how to build and deploy J2EE applications with SilverStream

Register and attend an educational seminar or event
http://www.silverstream.com/events

Sign up for training
http://www.silverstream.com/edu

Learn more about SilverStream's customers
http://www.silverstream.com/customers

General contact information
http://www.silverstream.com/contactus