Security FAQ

This FAQ addresses frequently asked questions about using Java EE Security within Sun GlassFish Application Server implementations. Additional resources can be found here. Please send any follow-up questions or comments to dev@glassfish.dev.java.net.

Q: What's new for security in Java EE 5?

- Java EE 5 requires Java SE 5 as the JVM platform.
- Java EE 5 supports security annotations for Java EE components (web/EJB/client module). This makes the Java EE component's deployment descriptor easier to write.

Q: Where can I get more information about Java EE security?

Java EE security is described in the Java EE 5 platform specification. More detailed information is also available from http://java.sun.com/j2ee/security. The Java EE 5 Tutorial is a good start place for

understanding Java EE security.

Q: How/why do I enable/disable the security manager? What is the impact on performance and Java EE authentication and authorization?

The Java security manager is disabled by default in the Sun Java System Application Server 9.0 PE edition. To enable/disable the security manager, you can either set the standard JVM system property `java.security.manager` or use the Admin Console. When the security manager is disabled, GlassFish will have better performance, but the JVM will not perform code-based security checks. It is important to note that even if the security manager is disabled, GlassFish still enforces Java EE standard authentication/authorization.

Q: I do not want to save GlassFish domain passwords in clear text. What should I do?

The GlassFish domain is password protected. Some CLI commands require the GlassFish domain administrator's password to operate. To make GlassFish secure by default, GlassFish has deprecated the CLI's "--password" option and added password alias support for CLI. Users can define the admin password alias in a disk file, and then use the CLI "--passwordfile" option to point to that disk file. For more detail information, please check the GlassFish Administration Guide.

How do I write/configure my own login module and plug it into GlassFish?

The GlassFish authentication subsystem is built upon realm and the standard Java JAAS framework - you can write your own realm and JAAS login module, and plug it into GlassFish. In the current implementation of GlassFish, your realm and JAAS module needs to be derived from `com.sun.appserv.security.AppservRealm` and `com.sun.appserv.security.AppservPasswordLoginModule` respectively. You can configure a realm and JAAS module by modifying the `config/domain.xml` file (adding your realm configuration information there) and `config/login.conf` file (adding your login module there) respectively.

Q: What is SSO (Single Sign-On) and how do I configure it in GlassFish?

SSO is an authentication solution allowing multiple applications to share the same user sign-on information.

- GlassFish has built-in support for SSO in its HTTP virtual server layer in a server instance. SSO in the HTTP virtual server layer is enabled by default. You can turn it off by defining the following property in domain.xml:
  ```
  <virtual-server ...>
      <property name="sso-enabled" value="false"/>
  </virtual-server>
  ```
- With the pluggable authentication support available in Glassfish, you can integrate it with external identity provider (such as openSSO) in two message layers - HttpServlet layer and SOAP layer. In the openSSO case, the framework enables applications deployed in GlassFish to utilize Federated SSO and across-domain Federation. For further information, please check openSSO in java.net.

Q: How do I configure SSL wire protocol in GlassFish?

You can configure the SSL protocol in the HTTP/SSL, RMI/SSL and IIOP/SSL layers through the GlassFish CLI or Admin Console. These interfaces allow you to define the SSL cipher suite, server certificate, and server or mutual authentication in an HTTPS connector. Inside GlassFish, the JMX

connector, which connects a JMX client and an MBean server, is based on RMI/SSL. By default, SSL is configured with a self-signed certificate and is ready to use for server authentication.

Q: How do I configure certificate-based client authentication?

The most common ways to configure client certificate authentication are in the HTTPS listener and in the web application deployment descriptor.

- For HTTPS listeners, you can set the attribute `client-auth-enabled="true"` for the `<ssl>` element in `<http-listener>`. Using this configuration, all requests through this listener will go through client certificate authentication.
- For web applications, you can configure the HTTPS listener's `<ssl>` attribute to `client-auth-enabled="false"` and, instead, configure the `<auth-method>` element in the web application's `web.xml` file to `CLIENT-CERT`. When this web application is accessed, the first request that qualifies for the `<auth-constraint>` will trigger the client-certificate authentication. The following example shows how to configure client-certificate authentcation in the `web.xml`.

```
<login-config>
    <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

Q: What's the Difference between a Server Authentication Module and a Pluggable Authentication Realm?

Both are externally developed security components that are integrated in the GlassFish server runtime.

Prior to invoking a Realm, the Glassfish runtime (typically) extracts a user name and password from the received request message. The runtime passes the user name and password through the Realm interface to the Realm implementation integrated at the pluggability point. The Realm implementation attempts to validate the password against it's repository, and on success, populates a JAAS subject with principals and credentials corresponding to the validated identity. In the Realm architecture, the Glassfish runtime, not the pluggable Realm is responsible for parsing the security information in the received message and extracting the information (that is, the user name and password) to be passed to the validation System (that is the Realm). The Realm interface is basically a pluggable password validation facility, which relies on the calling runtime to extract the username and password from the invocation message.

Server Authentication Modules are integrated in the Glassfish runtime such that the runtime passes the received request message (and the corresponding response message) across the pluggability interface to the Server Authentication Module integrated at the pluggability point. The authentication module is provided with access to the network messages, and it is expected to parse, validate, and modify their content as appropriate to the security mechanism implemented by the module. Like the realm implementation, the Server Authentication captures the results of a successful request message validation, by populating a JAAS Subject with principals and credentials corresponding to the authentication identity. Moreover, the message authentication SPI defines standard portable interfaces for password validation that can be used by an authentication module to invoke password validation functionality (when required by the authentication mechanism and not provided by the authentication module). Within Glassfish, the portable password validation interfaces are implemented as a veneer over the Realm pluggability point such that password validation is performed using the realm configured for the application.

Summary: Server Authentication Modules provide a standard, portable, pluggable network authentication facility. Glassfish Realms provide a pluggable password validation facility that is dependent on a message interpretation layer to extract the user name and password to be validated. Glassfish Realms are integrated within the GF implementation of the message authentication contract (that is, of JSR 196), such that when password validation is requested by a Server Authentication Modules, the Realm configured for the application is invoked to perform the validation.

Difference by Example: A server authentication module can implement FBL (Form Base Login), including constructing the redirects and validating the user name and password. The module could rely on a GF realm (e.g. A JDBC Realm) for the password validation. Conversely a GF Realm could not be expected to implement FBL.