Password Aliasing for EE 7

1. Introduction:

Best practices and common enterprise security policies dictate that we not store any passwords in clear text on the filesystem. There are a number of places where passwords are required in configuration, annotations and possibly even application code.

Password aliasing or indirection is a mechanism for storing and referencing a moniker or token instead of an actual clear text password. Resolving the token into an actual password for use at runtime is protected and only available to trusted code.

In order to support this in a portable way, Java EE 7 is standardizing a number of aspects of the solution. At the same time, the standard will not dictate the runtime implementation details for this support.

1.1. Standardized Aspects:

- Syntax for a password alias token
- Artifacts and annotations required to support aliasing
- Optional secure storage mechanism for bundling passwords with applications
    - it is assumed that the platform implementation will provide alias management capabilities in addition to support for this bundled credential store

1.2. Syntax:

The proposed syntax is based on what is available and in use by the RI/Glassfish today. It indicates that the value within the enveloping tag is an alias that needs to be resolved by the runtime to the actual password before use.

${ALIAS=*token*} within a known password element or annotation would represent such an alias.

Upon resolution, the runtime passes the clear text password to connections as is normally done. The password should be immediately discarded once it is no longer needed.

1.3. Affected Artifacts and Annotations:

- DataSourceDefinition annotation password element

Listing 1. DataSourceDefinition Annotation

```
@DataSourceDefinition(
          name="java:app/jdbc/test",
          className="com.mysql.jdbc.jdbc2.optional.MysqlDataSource",
          user="root",
          password="${ALIAS=password}",
          databaseName="test",
          serverName="localhost",
          portNumber=3306 )
```

- All similar annotations – need to be identified by individual expert groups

- the <password> element within the deployment descriptor <data-source> definition
- examples:

Listing 2. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
    <data-source>
        <name>java:app/env/testDS</name>
        <class-name>org.apache.derby.jdbc.ClientDriver</class-name>
        <server-name>localhost</server-name>
        <port-number>1527</port-number>
        <database-name>testDB</database-name>
        <user>APP</user>
        <password>${ALIAS=password}</password>
        <property>
            <name>connectionAttributes</name>
            <value>;create=true</value>
        </property>
    </data-source>
</web-app>
```

- Proprietary deployment descriptors for data-sources, resources or any other propreitary use of passwords
- possibly within the url of the elements of data-sources
- <env-entry-value> for password-valued environment entries

2. Secure Storage Format for Bundling Passwords with Applications

2.1. Introduction

In order to facilitate the provisioning of passwords and aliases at deployment time in a cross platform manner, we need to introduce a secure archive for bundling the alias and actual password values securely.

Platform vendors may use the deployed archive directly as the credential store for resolving password aliases or extract the contents of the artifact and populate a preferred credential store at deployment time for use by the platform runtime.

The details of the password store will be provided in a proposal dedicated to Improving Credential and SSL Configuration. That proposal will discuss a means to bundle these password stores as well as trust and keystores within applications and deploy them to a platform implementation.

3. Open Issues

3.1. File format for the standard EE password store

We need a standard format for securely exchange/deploy credentials to an EE platform. It is desired for this standard to be a well known industry standard that is interoperable and has sufficient support in tooling and consuming APIs.

Our primary candidate has been PKCS#12.

PKCS#12 is a "standard" format established and published by RSA Laboratories and is recognized as a defacto standard for Personal Information Exchange. Even though PKCS#12 is not a product of an actual industry standards body, RSA makes no patent claims on the general constructions within it.

There are a couple interesting issues with using PKCS#12 in general.

3.1.1. Interoperability

Given the open nature of much of the PKCS#12 specification there are numerous ways to represent the standard content and related metadata within a PKCS#12 file.

Interoperability would need to be addressed by clearly specifying the scheme to use in encoding the aliases, passwords and required metadata within PKCS#12.

It would also require enforcement within the TCK to ensure that the expected file format is consumed properly by the platform implementation.

3.1.2. Tooling

Currently there is a lack of generally available tooling for the management of passwords within the generic SecretBag constructs of PKCS#12.

JDK 7 support for PKCS#12 keystores is limited to private keys and certificate chains.

Support for secret keys and trusted certs will be added in JDK 8.

There is also planned enhancements to Keytool for management of secret keys within the JDK 8 timeframe.

Vendors can provide their own tooling that complies with the scheme defined within the standard use as described in the final specification.

3.1.3. Existing Providers

A number of providers have only implemented those portions of the PKCS#12 specification that is pertinent to existing PKI requirements and potentially entirely skip SecretBag entries within the stores while loading them.

Existing provider implementations will need to be aware of the specifics of the scheme used to represent EE password aliases and metadata in order to add support for them.

We would need to make a provider that is capabale of consuming this particular usage pattern of PKCS#12 generally available to platform implementations and tooling providers.

3.2. Password within password element of a URL

Whether we require the passord element within the data-source URL to be tokenized is a **minor** open issue that needs to be resolved.