RFP 123 - JAAS Integration

Confidential, Draft

8 Pages

Abstract

This document lays out requirements to support the full usage of the Java Authentication and Authorization Services (JAAS) inside an OSGi Framework.

0 Document Information

0.1 Table of Contents

0 Document Information 1
0.1 Table of Contents 1
0.2 Status 2
0.3 Acknowledgement 2
0.4 Terminology and Document Conventions 2
0.5 Revision History 2

1 Introduction **4**

2 Application Domain **5**

3 Problem Description **5**

4 Use case **6**
4.1 Support normative JAAS usages 6
4.1.1 Configuration 6
4.1.2 CallbackHandler 6
4.2 Obtain and/or Provision the Realm Definition using the Service Registry 7
4.3 Provision Realm Definitions from an OSGi Bundle 7
4.4 Offer an OSGi Platform-wide, Default Configuration Adaptation 7
4.5 Obtain Configuration from an OSGi Bundle 7
4.6 Integrate to the UserAdmin service for Authentication and Authorization 7

5 Requirements **8**

6 Document Support **9**
6.1 References 9
6.2 Author's Address 9
6.3 Acronyms and Abbreviations 10
6.4 End of Document 10

0.2 Status

This document suggests the following extension to the OSGi specification for the Open Services Gateway Initiative, and requests discussion. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial (0.1)	May 4, 2009	Jeff Trent, Oracle Corporation. jeff.trent@oracle.com

All Page Within This Box

Revision	Date	Comments
0.2	June 1, 2009	From comments received from gnodet@gmail.com (from Progress Corp). - Added notion of a Realm as a 1st class citizen within this RFP. - Replaced references of “application” with “realm”.
0.3	July 10, 2009	From comments received during today’s review session with Davanum M Srinivas <dims@us.ibm.com>, "Mihaylov, Dimitar" <dimitar.mihaylov@sap.com>, "Peshev, Peter" <peter.peshev@sap.com>, Guillaume Nodet <gnodet@progress.com>. Discussed the role of jsr-196 and jsr-115. All agreed that these are out of scope for this RFP. - We are also unanimously recommending to the EEG that work should begin on each one of these RFPs (115/Jacc & 196/AuthForContainers) - each deserving its own RFP. Volunteers anyone? * Make it more prominent in the document that the term 'Realm' will be used to replace the term 'Application' that developers may be accustomed to using. * Section 4: - Add (or amend existing) sub use case for - Dauth.login.defaultCallbackHandler - Add (or amend existing) sub use case for the jvm's security properties file (specifically login.config.url entries). * Section 4.2 & 4.3: - Remove "and LoginModules" from the description since LoginModules are not always thread safe and are not meant to be shared. - Change wording to be "obtain and/or provision a realm definition" instead of just "obtain". - Add wording to the affect that realm definitions can be accessed programmatically. Concern arose over the possibilities of rogue bundles adding realm definitions that will result in an escalation of privileges. This should be added as a note to the RFC writers somewhere in the document to use hooks, etc to avoid the situation.

Revision	Date	Comments
1.0	July 13, 2009	From comments received during today's review session with Davanum M Srinivas <dims@us.ibm.com>, "Peshev, Peter" <peter.peshev@sap.com>. - Removed use case 4.6 and requirement 12 – Coupling JAAS to UserAdmin was found to be problematic since there are no good choices in UserAdmin API for authentication. - Requirement 1.b – Only support the file-based usages of "-Djava.security.auth.login.config". - Requirement 2 – change wording to 'calling bundle' and remove the reference to manifest file since that dictates too much of the implementation strategy. - Requirement 4 – Reworded. - Requirement 5 – Remove the reference to a manifest file for similar reasons as specified above. - Requirements 6, 7, and 8 have been dropped (since they pertain to login modules accessed from the service registry – as per our last meeting this was deemed out of scope). - Requirement 9 (now req #6 in this document) – changed 'replaced' with the word 'modified'.

1 Introduction

The Java Authentication and Authorization Interface (JAAS) is an integral part of the Java platform that is widely used by a variety of application programming models. While the JAAS classes already can function inside an OSGi environment, JAAS is not suitably integrated with the OSGi architecture. In addition, the OSGi architecture enables a number of opportunities for improvement for use cases that a developer might expect when combining JAAS and OSGi.

All Page Within This Box

2 Application Domain

JAAS is available today in both the SE and EE environments and provides an extensible framework that abstracts the authentication and authorization mechanisms of an application or platform. Conceptually, JAAS has its roots in Unix's pluggable authentication module framework (PAM) established decades ago. Its extensibility means that different security mechanisms can be applied without modifying application-level code. The actual security mechanisms are pluggable via an SPI approach, where each plugin performs specific security logic pertaining to a particular kind of authentication process.

An application (i.e., a consuming osgi bundle), can integrate to JAAS for authentication via LoginContext and Configuration API's. The successful end-result of authentication is a `javax.security.auth.Subject` that is subsequently used for authorization decisions. The Subject is comprised of a set of Principals, each Principal representing a specific security identity attribute. It also may contain public and private credentials.

The SPI provider "plugin" for authentication processes is called the LoginModule. During the process of authentication a Subject is created (a.k.a. login) for the client by indirectly calling into one or more LoginModules. The JDK provides some LoginModule implementations out of the box (e.g., JndiLoginModule, Krb5LoginModule, etc.). Custom login modules can be written by a developer, and use of 3rd party login modules are also commonplace.

Henceforth within this document, the term "realm" will be used instead of "application" since "application" is an overloaded term and somewhat ambiguous to OSGi. A "realm" is identified by a [string] name property, and is related to a collection of one or more login modules [names] and flags used for those login modules (flags will be explained later in this document).

The authorization side of JAAS is already handled within the OSGi platform. When a SecurityManager is set permission checks are serviced by an OSGi-enabled policy provider. A bundle can provision permissions using the PermissionAdmin and the ConditionalPermissionAdmin services.

The primary focus of this RFP will concentrate mostly on the authentication side of JAAS.

3 Problem Description

The use of JAAS proliferated through the application space before OSGi became popular. While the API model is fairly adequate in an OSGi environment, the SPI model seems to be less so. For instance, a popular approach in the OSGi model for dealing with SPIs is using the whiteboard pattern in conjunction with the service registry. Consider the following stereotypical flow for authentication / login within a JAAS-enabled application:

```
LoginContext lc = new LoginContext("theContext");

lc.login();
```

All Page Within This Box

```
Subject sub = lc.getSubject();
```

Internally, the implementation engages the configured LoginModules appropriate for “theContext”-scoped LoginContext. In this example, “theContext” represents the realm name. Henceforth in this document, the term “Realm” will be used interchangeably with “Application”.

The traditional approach is to configure the set of login modules appropriate for “theContext” realm using a text file that is subsequently expected to be available at runtime through the client’s thread context classloader and at the time of the call.

Clearly, the use of SPI Services from the service registry, and avoiding usages of reflection or assumptions of the thread context classloader, would be a natural improvement when operating within the OSGi environment.

4 Use case

4.1 Support normative JAAS usages

This section applies to legacy JAAS code that is expected to work in an OSGi environment without modification.

4.1.1 Configuration

A user may wish to use traditional techniques for login, using code in their bundles resembling the following:

```
LoginContext lc = new LoginContext("theContext");

lc.login();

Subject sub = lc.getSubject();
```

According to the JAAS Specification, “The `LoginContext` consults a `Configuration` to determine the authentication services, or `LoginModule`**(s)**, configured for a particular application.”

In the above example, the Configuration parameter is omitted in the constructor to the LoginContext, resulting in the static Configuration.getConfiguration() method being called. Traditionally under the SE environment, the Configuration would be loaded from a file using a system property:

```
-Djava.security.auth.login.config=conf/loginmodules
```

4.1.2 CallbackHandler

Continuing with the example from 4.1.1 above, you will see that the CallbackHandler parameter was also omitted from the constructor. In this case, the JDK implementation makes use of the security property

```
-Dauth.login.defaultCallbackHandler=<callbackHandker>
```

All Page Within This Box

If specified, the JDK will call Class.forName() on that property to serve as the default CallbackHandler. This will occur from the caller's thread context at the time of the call.

The user expects that his legacy code will continue to operate normally within the OSGi environment. This also includes the *login.config.url* definitions from the security property file in the JVM lib directory.

4.2 Obtain and/or Provision the Realm Definition using the Service Registry

Traditionally in the SE environment, Realm definitions are configured via a file within a text file. For example:

```
ExampleRealm {

      MyRdbmsLoginModule required

      driver="a.path.to.the.Driver"

      url="jdbc:someurl…"

   };
```

A company adopting OSGi within their organization would like to deploy realm definitions directly to the Service Registry, preserving the same semantics as a file-based configuration, but using the more dynamic approach that the Service Registry has to offer.

4.3 Provision Realm Definitions from an OSGi Bundle

A user would like to have their OSGi bundle contain one or more Realm definitions and have them automatically provisioned to the Service Registry when the bundle is deployed. This is merely for a convenience mechanism so that a developer does not have to write code to perform the same.

4.4 Offer an OSGi Platform-wide, Default Configuration Adaptation

The Configuration implementation is responsible for loading and associating LoginModules to named contexts.

A company adopting OSGi within their organization would like a singleton Configuration to be implicitly available within the OSGi platform. The default Configuration would also be preconfigured with a default realm name

4.5 Obtain Configuration from an OSGi Bundle

A bundle deployer may wish to override the default, platform-wide Configuration with a bundle-specific Configuration. The expectation would be either to replace or augment the default, platform-wide Configuration.

The user wishing to do this has a specific Configuration in mind for his application, and does not want to make assumptions, or otherwise use, the global Configuration. Alternatively, the user may wish to apply custom login modules in addition to the set available within the platform for some exceptional cases unique to their applications. In this case a Configuration could be returned with another default realm name to use, or different flags for the login modules within the realm.

4.6 Integrate to the UserAdmin service for Authentication and Authorization

A strong motivation behind the UserAdmin service from OSGi 4.1 stemmed from the inability to use JAAS at the time of its writing due its requirements on the JDK version (see *User Admin Service* under *Reference*).

Additionally, the UserAdmin service offered more capabilities over and above standard JAAS with respect to a User and Group persistent repository.

Users have undoubted integrated to UserAdmin and are expecting the ability to continue to use it and not have to migrate to a "new" facility such as JAAS with more limited functionality. The overlapping features between UserAdmin and JAAS may be confusing to some users. Users will likely expect a clear relationship between JAAS and the UserAdmin service instead of a duplication of services.

The user is likely to expect seamless integration to the UserAdmin service for authentication and authorization if the UserAdmin service is available on the platform.

5 Requirements

1. Normative JAAS usages (this applies to legacy code that has not undergone any osgification):

   a. The static Configuration.setConfiguration() method MUST be injected with an appropriate implementation to insure proper integration to the OSGi platform. This MUST occur early enough during OSGi platform bootstrapping to guarantee that application bundles can make use of the Configuration during their activation phase. [Note: It is recommended that JAAS be started as an extension to the OSGi framework, but I'm not sure if saying "extension" is appropriate terminology to use in the RFP – this seems more appropriate language for the RFC.]

   b. It is RECOMMENDED the framework support a Configuration passed using the standard system property "`-Djava.security.auth.login.config`" to the OSGi framework when the configuration refers to a file. Attempt to use class-based usages SHOULD result in appropriate warning messages logged.

   c. Any LoginContext created without a CallbackHandler parameter SHOULD be supported, and SHOULD somehow seamlessly integrate to the OSGi platform. Similar constraints as above apply; the default CallbackHandler should be available during application's activation phase. This also includes a callback handler passed using the system property "`-Dauth.login.defaultCallbackHandler`".

2. It is RECOMMENDED to have the LoginContext's name default to a declared name from an artifact or attribute found within the calling bundle, or the bundle symbolic name itself. This is likely to necessitate wrapping the LoginContext by an OSGi-specific set of framework classes.

3. It is RECOMMENDED to have the LoginContext's CallbackHandler (if passed a null argument) default to a CallbackHandler declared within the bundle, and defaulting to a platform-wide default CallbackHandler if not specified by the bundle. Once again, this is likely to necessitate wrapping the LoginContext by an OSGi-specific set of framework classes.

4. The Realm definition SHOULD represent the relationship from a LoginContext's name to the set of login modules w/ flags for those login modules. These relationships SHOULD be exposed programmatically in some fashion within the OSGi platform.

All Page Within This Box

5. It is RECOMMENDED that there is a declarative way for loading a realm definition based on the contents of the bundle (elements, attributes, artifacts, etc.).

6. The OSGi JAAS integration SHOULD support a mechanism by which the platform-wide Configuration can be easily modified by a user's substitute configuration/implementation. It SHOULD support a programmatic substitution mechanism, provided that the bundle performing the replacement is privileged to do so.

7. The OSGi JAAS integration SHOULD support a bundle-specific override Configuration or Realm definition.

8. The OSGi JAAS integration MUST be secure when a SecurityManager is enabled. Permissions MUST guard:

   a. Configuration placed into the global context.

   b. Default CallbackHandler placed into the global context.

   c. Realm definitions placed into the global context [special consideration should be considered to prevent any escalation of privileges].

6 Document Support

6.1 References

[1] Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2] Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0.

[3] User Admin Service, OSGi Service Platform Release 4.1, April 2007.

[4] Benjamin Reed, Permission Admin Service, OSGi Service Core Platform 1.0, April 2001.

[5] JAAS Reference Guide, http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html

6.2 Author's Address

All Page Within This Box

Name	Jeff Trent
Company	Oracle Corporation
Address	330 Fellowship Road, Suite 100 – Moorestown, NJ 08054
Voice	(856) 359-2937
e-mail	jeff.trent@oracle.com

6.3 Acronyms and Abbreviations

JAAS: Java Authentication and Authorization Service

JCP: Java Community Process

LM: Login Module

6.4 End of Document

All Page Within This Box