RFC 73: Permission Update

Final

14 Pages

Abstract

Permission Admin is used to assign a set of permissions to bundles. This RFC describes changes to the OSGi specifications to create finer grained permissions. The current AdminPermission is too broad and needs to be broken up and additional permissions are necessary for new privileged function.

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 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.`

0.3 Revision History

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

Revision	Date	Comments
Initial	NOV 28 2003	Initial Revision. Benjamin C Reed, IBM, breed@almaden.ibm.com
0.1	JAN 22 2004	Small changes from meeting & addition of zones, Benjamin Reed
0.2	FEB 16 2004	Allow recursion for permission sets. Use location table to implement permission sets. Other small fixes. Benjamin Reed
0.3	APR 22, 2004	Added predefined zones, added permission scoping by manifest, added permission.perm. Benjamin Reed
0.4	May 28, 2004	Removed permission sets. Added some more detail to permissions.perm file. Took a swipe at restricting installation based on signatures.
0.5	29 May 2004	Update to move AdminPermission changes into this RFC. BJ Hargrave, IBM, hargrave@us.ibm.com
0.6	08 Jun 2004	Clarification of permission evaluation when calling setPermissions. BR.
0.7	22 Jul 2004	Added MissingPermissionListener and query methods to find bundles matching qualifiers. BR.
0.8	03 Oct 2004	Removed Signer language (went to RFC 95). Clarified filter use in AdminPermission.
0.9	01 Dec 2004	Fix the Configuration fine grained permission.
0.10	14 Jan 2005	Made ConfigurationPermission the only permission used by ConfigurationAdmin. Clarified PermissionAdmin a bit. Simplified the permission evaluation for PermissionAdmin.setPermssions. Simplified signer specification in name parameter of AdminPermission and added reference to RFC 95.
0.11	31 January 2005	Changed AdminPermission constructor to take Bundle object instead of Integer. Added notes on how the permission checks are done on installBundle, uninstall, and update. Moved BundlePermission to RFC 70.
0.12	28 February 2005	Added extensionLifecycle action.

Final	27 May 2005	No changes BJ Hargrave, hargrave@us.ibm.com

1 Introduction

The OSGi Service Platform R3 specification optionally supports a secure environment by enforcing Java 2 permissions. The R3 spec also defines specific permissions for access OSGi specific functions. Post R3 requirements and related design activity have made it clear that the use of AdminPermission needs improvement to define a finer grained permission structure.

2 Application Domain

The OSGi SP R3 spec defines the following Framework permissions:

- AdminPermission – Enables access to the administrative functions of the Framework.
- ServicePermission – Controls service object registration and access.
- PackagePermission – Controls importing and exporting packages.
- Framework implementation defined resource permission – Enables a bundle to access its own resources

The R3 spec also defines UserAdminPermission and WirePermission for use by the UserAdmin and WireAdmin services respectively

2.1 Administrative Operations

The AdminPermission is widely used within the specification for many different privileged operations which have been grouped together as administrative in nature. The following table lists all the methods in the R3 spec which require the caller to have AdminPermission.

R3 methods requiring AdminPermission
Bundle.getHeaders

All Page Within This Box

R3 methods requiring AdminPermission
Bundle.getLocation
Bundle.getResource
Bundle.stop
Bundle.uninstall
Bundle.update
Bundle.start
BundleContext.installBundle
BundleContext.addBundleListener for SynchronousBundleListener
BundleContext.removeBundleListener for SynchronousBundleListener
Configuration.setBundleLocation
Configuration.getBundleLocation
ConfigurationAdmin.createFactoryConfiguration
ConfigurationAdmin.getConfiguration
ConfigurationAdmin.listConfigurations
PackageAdmin.refreshPackages
PermissionAdmin.setPermissions
PermissionAdmin.setDefaultPermissions
StartLevel.setStartLevel
StartLevel.setBundleStartLevel
StartLevel.setInitialBundleStartLevel

3 Problem Description

The current AdminPermission usage in the OSGi specifications is too coarse grained. Many different aspects of administration are secured by the single AdminPermission. AdminPermission is not parameterizable. We must extend and enhance the OSGi permissions to allow different bundles to perform only a select subset of the privileged administrative operations. This will allow an appropriate separation of administrative concerns to be made and prevent the need to assign overly broad permissions.

For example:

An HttpService implementation must be given AdminPermission to support the use of the default HttpContext. This is necessary to allow the default HttpContext to call Bundle.getResource to access resources in the servlet bundles.

A bundle wishing to administer configurations via ConfigurationAdmin must be given AdminPermission to create or change a configuration.

Thus bundles are given significantly more permissions than necessary to accomplish their specific, limited needs.

4 Requirements

We need to modify the permissions such that bundles are only given the minimal permission to perform the specific privileged operations they need.

We should also attempt to support legacy bundles without requiring code changes or permission configuration changes.

The system should remain simple enough for a normal operator to manage.

5 Technical Solution

5.1 AdminPermission Changes

The proposed solution is to modify the AdminPermission to take parameters to allow finer grained assignment and testing of administrative permission. This will be used for those privileged operations currently defined in the R3 spec and newly defined privileged operations that closely map to R3.

AdminPermission will be updated to define an name and action parameters. The name parameter of the AdminPermission will be the bundle id to which the permission applies. The actions will cover specific subset of privileged administrative operations.

5.1.1 Name Parameter

AdminPermission will have an additional constructor which will take a Bundle object as the name parameter. This will allow an AdminPermission to be created directly from a Bundle object. Apart from the default constructor, this is the only means to construct an AdminPermission for use in calling SecurityManager.checkPermission. AdminPermission objects constructed with the default constructor have special handling described in 5.1.6.

The constructor that takes a String for the name is only used to assign an AdminPermission to a bundle using PermissionAdmin. See section 5.1.4 for further explanation.

5.1.2 Actions Parameter

The actions parameter of AdminPermission will specify the subset of privileged administrative operations the permission addresses. The complete set of privileged operations addressed by the R3 version of AdminPermission will be partitioned into a set of actions.

Action	**Privileged operations**
metadata	Bundle.getHeaders Bundle.getLocation
resource	Bundle.getResource Bundle.getEntry Bundle.getEntryPaths Bundle resource/entry URL creation
class	Bundle.loadClass

All Page Within This Box

Action	Privileged operations
lifecycle	BundleContext.installBundle Bundle.update Bundle.uninstall
extensionLifecycle	BundleContext.installBundle Bundle.update Bundle.uninstall
execute	Bundle.start Bundle.stop StartLevel.setBundleStartLevel
listener	BundleContext.addBundleListener for SynchronousBundleListener BundleContext.removeBundleListener for SynchronousBundleListener
permission	PermissionAdmin.setPermissions PermissionAdmin.setDefaultPermissions
resolve	PackageAdmin.refreshPackages PackageAdmin.resolveBundles
startlevel	StartLevel.setStartLevel StartLevel.setInitialBundleStartLevel

The special action "*" represents all actions.

5.1.3 Method Notes

The following methods have special notes.

Method	Note
PermissionAdmin.setPermissions	Requires AdminPermission(systemBundle,"permission"). The name value must be the system Bundle since this method affects the entire framework.
PermissionAdmin.setDefaultPermissions	Requires AdminPermission(systemBundle,"permission"). The name value must be the system Bundle since this method affects the entire framework.

All Page Within This Box

Method	Note
PackageAdmin.refreshPackages	Requires AdminPermission(systemBundle,”resolve”). The name value must be the system Bundle since this method affects the entire framework.
PackageAdmin.resolveBundles	Requires AdminPermission(systemBundle, ”resolve”). The name value must be the system Bundle since this method affects the entire framework.
StartLevel.setStartLevel	Requires AdminPermission(systemBundle, ”startlevel”). The name value must be the system Bundle since this method affects the entire framework.
StartLevel.setInitialBundleStartLevel	Requires AdminPermission(system Bundle, ”startlevel”). The name value must the system Bundle since this method affects the entire framework.
BundleContext.installBundle	Requires the caller to have an AdminPermission assigned whose name parameter filter expression can be satisfied by the bundle being installed and an action of “lifecycle”*. When and how this check happens is implementation specific, but it must be done before the bundle is actually installed. Normally the check will happen right after the bundle’s Manifest and signature files have been downloaded.
Bundle.uninstall	Requires the caller to have an AdminPermission assigned whose name parameter filter expression can be satisfied by the bundle being uninstalled and an action of “lifecycle”*.
Bundle.update	From a permission check point of view, Bundle.update is modeled as uninstall followed by installBundle. First, the framework does a permission check to uninstall the old version of the bundle using the Bundle.uninstall logic noted above. Then, the framework checks for permission to install the new version of the bundle according to the installBundle logic noted above. If either check fails, Bundle.update fails with a SecurityException.

* If the bundle being installed/updated/uninstalled is an extension bundle, the action must be “extensionLifecycle” rather than “lifecycle”.

5.1.4 Setting AdminPermission via PermissionAdmin

When assigning permissions to be bundle via PermissionAdmin, it is difficult for the administrator to have *a priori* knowledge of the bundle id that is assigned to a bundle. Since bundle ids are assigned locally by the framework during install of a bundle, a given bundle may have a unique id on every framework upon which it is installed. Thus we need a mechanism for the administrator to use to select the appropriate bundle so that AdminPermissions can be created for a bundle’s ProtectionDomain.

In order to provide a more flexible way to specify the bundle for an AdminPermission, PermissionAdmin will provide special support for assigning AdminPermissions. When a PermissionInfo with an AdminPermission type is specified to PermissionAdmin, the name parameter of the PermissionInfo must be a filter string. The filter string

All Page Within This Box

can only be used with PermissionInfo. If a bundle constructs an AdminPermission object using a filter string, it will never be implied by any other AdminPermission or AdminPermission collection.

The supported filter string keys are:

Key	Description
Id	The value of this key must be a long specifying the id of a bundle.
location	The value of this key must be a String specifying the location of a bundle. Filter wildcards for Strings are supported allowing the value to specify a set of bundles. The wildcards are evaluated according to java.io.FilePermission's wildcard expansion rules.
signer	The value of this key must be an expression to match a signature chain as specified in RFC 95 [5].
Name	The value of this key must be a String specifying the symbolic name of a bundle. Filter wildcards for Strings are supported allowing the value to specify a set of bundles. A single symbolic name may also map to a set of bundles.

The special name "*" represents all bundles.

5.1.5 ConfigurationPermission

To allow fine grained permissions to ConfigurationAdmin, the following methods need to be altered:

Configuration.setBundleLocation

Configuration.delete

Configuration.update

ConfigurationAdmin.createFactoryConfiguration

ConfigurationAdmin.getConfiguration

ConfigurationAdmin.listConfigurations

These methods will be changed to use ConfigurationPermission. The name parameter of the ConfigurationPermission indicates the pids that the permission applies to. The pids can be specified using wildcard character as a suffix. The action is either "get" or "set". The "set" action is needed for the three Configuration methods and ConfigurationAdmin.createFactoryConfiguration. The "get" action is needed for ConfigurationAdmin.getConfiguration. Also, ConfigurationAdmin.listConfigurations will only return Configurations to which the caller has "get" access.

5.1.6 Backwards Compatibility

To provide backwards compatibility for existing code, we define the no argument constructor for AdminPermission to be the same as constructing AdminPermission with a name argument of "*" and an actions argument of "*".

All Page Within This Box

This results in an AdminPermission which applies to all bundle id and all actions. The no argument constructor will be deprecated to indicate that the 2 argument constructor should now be used. However, legacy bundles will still use the no argument constructor. Thus, an AdminPermission constructed with the no argument constructor will be implied by an AdminPermission with the name and action of "*".

Each bundle must be given AdminPermission(<bundle id>, "resource") so that it can access it's own resources. This is an implicit permission that must be given to all bundles. This replaces the framework implementation defined permission in R3 4.7.1

6 API Javadoc

6.1 Class ConfigurationPermission

public class **ConfigurationPermission**
extends java.security.BasicPermission

This permission controls access to ConfigurationAdmin. It enables access to pids that may be described using wildcards as explain in BasicPermission. It also controls the type of access allowed. "get" access will allow only queries. "set" access allows configurations to be added, changed, and deleted.

See Also:
Serialized Form

Field Summary	
`static java.lang.String`	**`GET_ACTION`** The action string for the `GET_ACTION` action: value is "get"
`static java.lang.String`	**`SET_ACTION`** The action string for the `SET_ACTION` action: value is "set"

Constructor Summary
`ConfigurationPermission``(java.lang.String name, java.lang.String action)` Constructs a `ConfigurationPermission` with the given name and action.

Method Summary	
`boolean`	**`implies`**`(java.security.Permission perm)` This method returns true if the name of `perm` is implied by this permission according to

All Page Within This Box

	`BasicPermission.implies` and any actions of `perm` are also actions of this permission.

Field Detail

6.1.1 GET_ACTION

```
public static final java.lang.String GET_ACTION
```

The action string for the `GET_ACTION` action: value is "get"

See Also:

Constant Field Values

6.1.2 SET_ACTION

```
public static final java.lang.String SET_ACTION
```

The action string for the `SET_ACTION` action: value is "set"

See Also:

Constant Field Values

Constructor Detail

6.1.3 ConfigurationPermission

```
public ConfigurationPermission(java.lang.String name,
                               java.lang.String action)
```

Constructs a `ConfigurationPermission` with the given name and action. The name describes the pids that can be accessed using this permission and the action indications whether than can be queried and or set.

Parameters:

`name` - the pids that can be accessed. Can use wild cards as described in BasicPermission.

`action` - the type of access allowed: get and/or set.

See Also:

`BasicPermission`

Method Detail

6.1.4 implies

```
public boolean implies(java.security.Permission perm)
```

This method returns true if the name of `perm` is implied by this permission according to `BasicPermission.implies` and any actions of `perm` are also actions of this permission.

Overrides:

`implies` in class `java.security.BasicPermission`

See Also:

`BasicPermission.implies(java.security.Permission)`

All Page Within This Box

7 Considered Alternatives

We had considered using a configure action on AdminPermission rather than introducing ConfigurationPermission. This proved to be problematic since the target of configure is a PID not a bundle.

8 Security Considerations

This RFC specifies major changes to the AdminPermission which is used to secure privileged administrative operations for the framework. Thus it is critical to the security of the overall system that the design AND implementation of AdminPermission be correct.

9 Document Support

9.1 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]. Li Gong, Gary Ellison, and Mary Dageforde, Inside Java 2 Platform Security: Architecture, API Design, and Implementation (2nd Edition), Addison-Wesley, March 2003.

[4]. RFP 48 Security Enhancements, http://membercvs.osgi.org/rfps/rfp-0048-Security%20Enhancements.doc

[5]. RFC 95 Conditional Permissions, http://membercvs.osgi.org/rfcs/rfc0095/rfc0095.doc

9.2 Author's Address

All Page Within This Box

Name	Benjamin Reed
Company	IBM
Address	650 Harry Rd, San Jose, CA 95037, USA
Voice	(408) 927-1811
e-mail	breed@almaden.ibm.com

Name	BJ Hargrave
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 8838
e-mail	hargrave@us.ibm.com

9.3 Acronyms and Abbreviations

9.4 End of Document

All Page Within This Box