JMX 1.2 Security

Access Control for the MBeanServer and its MBeans

Introduction

This proposal extends the current JMX specification with features that allow administrators to control access to the MBeanServer and individual MBean APIs.

In addition to adding support for access control, it is an explicit goal of this proposal not to invent any new security mechanisms. Only the mechanisms provided by the Java 2 platform security architecture and the Java Authentication Authorization Service (JAAS) will be leveraged by this proposal.

JMX Security Exposures

Identifying the risks, or exposures, is the first step in the design of any security solution. This section describes the JMX exposures this proposal addresses.

1) *MBeanServer API Access*. The core component of the JMX architecture at the agent level is the MBeanServer. The MBeanServer provides access to all of the MBeans registered with it. The MBeanServer provides additional management functionality such as the creation of MBean instances and the management of MBeanServer notifications. JMX 1.1 defined the permissions needed to access MBeanServer instances; this proposal will impact and extend this part.

2) *MBean API Access*. MBeans are the mechanism for monitoring and manipulating managed resources. An MBean allows a client to get and set resources attributes and to affect the resources behavior via method invocations. In the current specification none of these activities is secured. When MBeans are accessed through the MBeanServer this access must be secured.

3) *Trusted MBean Sources*. It is important that only MBeans from trusted codebases can be registered with the MBeanServer. If MBeans from arbitrary codebases are allowed to register it would be easy for a hostile MBean to spoof another MBean and exercise unauthorized control over managed resources.

JMX Permissions

This proposal addresses the exposures described above by defining new permission types. A Java permission, `java.security.Permission` or one of its subclasses, represents the authority to access some resource or to perform some operation. This is exactly what is needed to address the MBeanServerFactory API and MBeanServer API.

MBeanServerPermission

```
public final class MBeanServerPermission
extends java.security.Permission
```

This class represents the authority to access MBeanServer instances.

The MBeanServerPermission contains a target name or a comma-separated list of target names but no actions list; you either have the named permission or you don't.Only the null value or the empty string are allowed for the action to allow the policy object to create the permissions specified in the policy file.

The target name is the name of the operation you invoke to access MBeanServer instances.

The following table lists all the possible MBeanServerPermission target names and for each provides a description of what the permission allows and a discussion of the risks of granting the permission.

Permission Target Name	What the Permission allows	Associated Risks
newMBeanServer	Creating new MBeanServer instances of MBeanServer not registered with the MBeanServerFactory	Malicious code can create MBeanServer instances.
createMBeanServer	Creating and registering an MBeanServer with the MBeanServerFactory	Malicious code can create and register MBeanServer instances.
findMBeanServer	Looking up an MBeanServer instance registered with the MBeanServerFactory	Malicious code can obtain a reference to existing MBeanServer instances.
releaseMBeanServer	Deregistering an MBeanServer from the MBeanServerFactory	Malicious code can remove existing MBeanServer instances
getMBeanServerBuilder	Retrieve the MBeanServerBuilder used to create MBeanServer instances	Malicious code can obtain a reference on the MBeanServerBuilder
setMBeanServerBuilder	Set the MBeanServerBuilder used to create MBeanServer instances	Malicious code can replace the MBeanServerBuilder
*	All the above	All the above

Implementations should also prevent clients from instantiating concrete MBeanServer implementation objects, *e.g.*, via `new com.sun.jdmk.MBeanServerImpl()`.

This proposal will slightly extend the meaning of the “createMBeanServer” target name. When MBeanServerPermission “createMBeanServer” is granted, then MBeanServerPermission “newMBeanServer” is implied.

Target names can be specified in a comma separated list.

Example Policy

The simplest MBeanServer access policy is to grant all signers and codebases the permission to create and access MBeanServer instances:

```
grant
{
   permission javax.management.MBeanServerPermission “*”;
};
```

Here is a more restrictive policy that grants *ApplicationOne* contained in applone.jar the permission to create and release MBeanServers, but not to find them:

```
grant applone.jar
{
   permission javax.management.MBeanServerPermission
“createMBeanServer, releaseMBeanServer”;
};
```

Here is a policy that grants *ApplicationTwo* contained in appltwo.jar the permission to find MBeanServers:

```
grant appltwo.jar
{
   permission javax.management.MBeanServerPermission
“findMBeanServer”;
};
```

MBeanPermission

```
public final class MBeanPermission
extends java.security.Permission
```

This class represents access to MBeanServer operations.

The MBeanPermission contains a target name and an action or a comma-separeted list of actions.

The target name is of the form `"classname#member[objectname]"`:

- the *classname* denotes the java classname of the MBean.

  The MBean classname is obtained using the `MBeanInfo.getClassName()` method invoked on the MBeanInfo object of the specified MBean.

  The classname can use wildcards following the same wildcard usage as the one defined by the java.security.BasicPermission for target names.

- the *member* denotes the attribute name for a "get/setAttribute" and the operation name for an "invoke".

  The member can use wildcards "*" to denote any attribute/operation in the MBean.

- the *objectname* denotes the ObjectName of the MBean subject of this operation.

  The objectname can use wildcards following the JMX rules for ObjectName patterns.

The rules for target names are the following:

- each target name part (classname, member and objectname) is optional.
- when evaluating HAVE.implies(NEED) for the classname part:
  - if HAVE omits the classname the result is true (this means that omitting the classname in the policy file is equivalent to supplying "*").
  - if NEED omits the classname the result is true (this means that the classname is not relevant to the permission being checked).
- when evaluating HAVE.implies(NEED) for the member part:
  - if HAVE omits the member the result is true (this means that omitting the member in the policy file is equivalent to supplying "#*").
  - if NEED omits the member the result is true (this means that the member is not relevant to the permission being checked).
- when evaluating HAVE.implies(NEED) for the objectname part:
  - if HAVE omits the objectname the result is true (this means that omitting the objectname in the policy file is equivalent to supplying "[*:*]").

- if NEED omits the objectname the result is true (this means that the objectname is not relevant to the permission being checked).

- the evaluation of the HAVE.implies(NEED) for the three target name parts (classname, member and objectname) must be true for the overall target name 'implies' to be true.

The actions list is a comma-separated list of the MBeanServer methods listed below or "*" to denote any method. The actions list cannot be either null or the empty string.

The following table lists all the possible MBeanPermission actions and for each provides a description of what the permission allows and a discussion of the risks of granting the permission.

Permission Action Name	What the Permission allows	Associated Risks
addNotificationListener	To add notification listeners to specified MBeans that can emit notifications	Listeners, filters and handbacks can contain malicious code
getAttribute	To get the value of an attribute of an MBean	Malicious code can find out the value of security-sensitive attributes
getClassLoader	To get the classloader reference represented by the named MBean	Malicious code can retrieve a reference on a classloader and add new classes to it
getClassLoaderFor	To get the ClassLoader that was used for loading the class of the named MBean.	Malicious code can retrieve a reference on a classloader and add new classes to it
getClassLoaderRepository	To get a reference on the MBeanServer's classloader repository	Malicious code can retrieve a reference on the classloader repository of the MBeanServer and use it to load classes
getMBeanInfo	To get the MBeanInfo of the named MBeans	Malicious code can find out the classname, constructors, operations, attributes and notifications of the MBean
getObjectInstance	To get the ObjectInstance of the named MBeans	Malicious code can find out the classname of the MBean
instantiate	To instantiate java classes or MBeans	Malicious code can add new classes to the JVM and retrieve MBean object references
invoke	To invoke operations on an MBean	Malicious code can invoke operations on the MBean (for example start/stop a service, etc)
isInstanceOf	To know if the MBean is of a certain class	Malicious code can find out the class type of the MBean
isRegistered	To know if an MBean is registered with a certain ObjectName	Malicious code can find out if an MBean with a certain ObjectName is registered
queryMBeans	To query the MBeanServer for registered MBeans	Malicious code can find out the MBeans registered in the MBeanServer
queryNames	To query the MBeanServer for registered MBeans	Malicious code can find out the MBeans registered in the MBeanServer

Permission Action Name	What the Permission allows	Associated Risks
registerMBean	To register MBean instances	The registered MBean can contain malicious code
removeNotificationListener	To remove listeners from MBeans that can emit notifications	Modify the behavior of MBeans that registered as listeners of another MBeans
setAttribute	To set the value of an attribute of an MBean	Malicious code can modify the value of security-sensitive attributes
unregisterMBean	To unregister MBeans	Modify the behavior of applications that rely on the presence of MBeans
*	All the above	All the above

There are MBeanServer operations that are not subject to any permission check, namely `getDefaultDomain` and `getMBeanCount`.

This proposal will slightly extend the meaning of the "queryMBeans" target name. When the MBeanPermission `"queryMBeans"` is granted, then the MBeanPermission `"queryNames"` is implied.

The target name assumes a different (but intuitive) meaning when the action is queryMBeans or queryNames. Instead of representing the ObjectName of the MBean(s) on which the user is allowed to call the named action (like "getObjectInstance" or "isInstanceOf"), the target name represents a "security filter" that filters out from the returning set the ObjectNames of the MBeans that do not match in terms of classname and objectname (see example below, and the implementation notes at the end).

Access to these MBeanServer operations is granted if the user has access to the MBeanServer instance.

As shown below, certain actions require the presence of the operation or attribute name (eventually implied by the presence of a wildcard), while others do not require it.

For example, let's suppose to have the following MBean class, registered in the MBeanServer under ObjectName "domain:key=value".

```
package net.jmx;

public interface FooMBean
{
    public int getBar();
    public void setBar(int bar);
    public void doIt();
}

public class Foo implements FooMBean {...}
```

The target name of the "Bar" attribute is: `net.jmx.Foo#Bar`, or `net.jmx.Foo#Bar[domain:key=value]`.

The target name of the "doIt" operation is: `net.jmx.Foo#doIt`, or `net.jmx.Foo#doIt[domain:key=value]`.

No distinction is made between overloaded operations:

```
package net.jmx;

public interface FooMBean
{
    public int getBar();
    public void setBar(int bar);
    public void doIt();
    public void doIt(int param);
}
public class Foo implements FooMBean {...}
```

In this case `net.jmx.Foo#doIt` refers to both `Foo.doIt()`and to `Foo.doIt(int param)`.

Attribute and operation names may be wildcarded. So, `net.jmx.Foo#*` designates all of Foo's attributes and operations.

classnames may also be wildcarded. So, `net.jmx.*` designates all operations and attributes of all MBeans whose fully qualified classname starts with `net.jmx`.

In the rare case of MBeans that are inner classes, the syntax would be similar to this one: `net.jmx.Outer$Inner#*`. It is not allowed to wildcard inner classes; the following syntax (or any other variation) is thus illegal: `net.jmx.Outer$*`.

The ObjectName specified in square brackets must be a valid complete ObjectName or a valid ObjectName pattern.

The target name `net.jmx.Foo#doIt[*:*]` is equivalent to `net.jmx.Foo#doIt`.

The target name `net.jmx.Foo#*[confidential:*]` refers to all operations and attributes of the `net.jmx.Foo` MBean instances registered under the "`confidential`" domain with any property.

The getAttributes/setAttributes MBeanServer operations are protected by the individual getAttribute/setAttribute operations. They must be checked and filtered out in a similar way as the queryMBeans/queryNames operations are:

1. Check that the caller has the rights to getAttribute ignoring the member.

   The MBeanServer would check for:

   MBeanPermission("myclass[d:k=v]", "getAttribute").

2. If the caller does not have the right, throw a SecurityException.

3. For each attribute in the attribute list:

   * check that the caller has the right to get/set it

   The MBeanserver would check for,

MBeanPermission(“myclass#attr*n*[d:k=v]”, “getAttribute)

attr*n* ---> n = 1..number of attributes

* if the caller does not have the right to get/set it remove the attribute from the list

4. Invoke the get/setAttributes operation on the MBeanServer with the reduced list of attributes for which the caller has the right to perform the get/setAttribute operation.

5. Return the result of the operation.

Two MBeanPermissions must be granted to the caller in order to perform the createMBean() MBeanServer operations: “instantiate” and “registerMBean”. This sounds correct because the createMBean operation first instantiates the MBean class and then registers the Mbean.

The “registerMBean” action is checked in both the createMBean and registerMBean MBeanServer operations in two steps. This is due to the user being able to provide a null ObjectName in the call and the real ObjectName in the MBean's preRegister method.

1. First, the MBeanServer checks that the caller has the right to invoke the “registerMBean” operation with the ObjectName provided by the user in the call (or omitted if the user provided null).

   MBeanPermission(“classname[objectname]”, “registerMBean”) or

   MBeanPermission(“classname”, “registerMBean”) if objectname is null.

2. Second, the MBeanServer checks that the caller has the right to invoke the “registerMBean” operation with the ObjectName returned by the preRegister method in the MBean.

   MBeanPermission(“classname[objectname_from_preRegister]”, “registerMBean”)

The deserialize() MBeanServer operations are covered by the “getClassLoader”, “getClassLoaderFor” and the “getClassLoaderRepository” actions depending on the deserialize method being invoked. This sounds correct because the deserialize operations will be implemented using these methods.

```
public ObjectInputStream deserialize(ObjectName name,
                                     byte[] data)
```

will be coverd by the action “getClassLoaderFor”.

```
public ObjectInputStream deserialize(String className,
                                     byte[] data)
```

will be coverd by the action “getClassLoaderRepository”.

```
public ObjectInputStream deserialize(String className,
                                     ObjectName loaderName,
                                     byte[] data)
```

will be coverd by the action “getClassLoader”.

Example Policy

The simplest MBean access policy is to grant all signers and codebases access to all MBeans:

```
grant
{
   permission javax.management.MBeanPermission “*”, “*”;
};
```

Here is a more restrictive policy that grants *ApplicationOne* contained in applone.jar the permission to get the MBeanServer's classloader repository:

```
grant applone.jar
{
   permission javax.management.MBeanPermission “”,
“getClassLoaderRepository”;
};
```

Here is a policy that grants *ApplicationTwo* contained in appltwo.jar the permission to call the isInstanceOf and getObjectInstance operations for MBeans from any class but registered under the “d1” domain:

```
grant appltwo.jar
{
   permission javax.management.MBeanPermission “[d1:*]”,
“isInstanceOf, getObjectInstance”;
};
```

Here is a policy that grants *ApplicationThree* contained in applthree.jar the permission to find MBeanServers and to call the queryNames operation restricting the returned set to only the MBeans under the “JMImplementation” domain:

```
grant applthree.jar
{
   permission javax.management.MBeanServerPermission
“findMBeanServer”;
   permission javax.management.MBeanPermission “JMImplementation:
*”, “queryNames”;
};
```

Let’s suppose that an MBeanServer accessible by ApplicationThree has 3 MBeans registered under these ObjectNames:

JMImplementation:type=MBeanServerDelegate

:mbean=default

Domain:key=value

Then ApplicationThree can query the MBeanServer (since it has an MBeanPermission with “queryNames” action):

```
MBeanServer server =
    MBeanServerFactory.findMBeanServer(null).get(0);
Set mbeans = server.queryNames(null, null);
```

The returned set only contains one ObjectName element, namely “JMImplementation: type=MBeanServerDelegate”, since:

The client has an MBeanPermission with a “queryNames” action, so the client is allowed to call MBeanServer.queryNames()

There is only one MBean whose ObjectName matches the ObjectName pattern specified in the MBeanPermission granted in the policy file.

Any MBeanPermission with a “queryNames” action will allow ApplicationThree to call MBeanServer.queryNames(); if an MBeanPermission with a “queryNames” action is not granted, a SecurityException is thrown. Otherwise, the ObjectName specified as target name in the permission will filter out the returning set of ObjectNames that do not match.

The same applies to queryMBeans: the returning set will contain only ObjectInstances whose ObjectName matches the specified permissions.

Here is a more restrictive policy that grants *ApplicationOne* permission to create and manipulate the Foo MBean:

```
grant applone.jar
{
   permission javax.management.MBeanPermission “net.jmx.Foo”,
“instantiate, registerMBean”;
   permission javax.management.MBeanPermission
“net.jmx.Foo#doIt”, “invoke, addNotificationListener,
removeNotificationListener”;
};
```

The first permission ignores the objectname. The operation or attribute name is not required by these two actions.

The second permission, however, uses the member part for the “invoke” action and ignores it for the “add/removeNotificationListener” actions.

Here’s one that grants *ApplicationTwo* permission to listen for notifications on MBeans whose classname are contained in the java package “net.jmx”:

```
grant appltwo.jar
```

```
{
   permission javax.management.MBeanPermission “net.jmx.*”,
“addNotificationListener, removeNotificationListener”;
};
```

The example below does not allow *ApplicationThree* to access the Foo MBean, since the “setAttribute” action is not granted, the “getAttribute” action cannot be applied to MBean operations, and the “invoke” action cannot be applied to MBean attributes:

```
grant applthree.jar
{
  permission javax.management.MBeanPermission “net.jmx.Foo#doIt”,
“getAttribute”;
  permission javax.management.MBeanPermission “net.jmx.Foo#Bar”,
“invoke”;
};
```

If the caller performs a getAttributes call in the MBeanServer the call will return an empty set. The caller has the right to invoke getAttribute but none of the attributes in the list can match the member part of the permission in the policy file.

MBeanTrustPermission

```
public final class MBeanTrustPermission
extends java.security.BasicPermission
```

This permission represents “trust” in a signer or codebase. MBeanTrustPermission contains a target name but no actions list. A single target name, “register”, is defined for this permission. Only the null value or the empty string are allowed for the action to allow the policy object to create the permissions specified in the policy file.

If a signer, or codesource is granted this permission, then it is considered a trusted source for MBeans. Only MBeans from trusted sources may be registered in the MBeanServer.

Example Policy

The simplest policy for MBean sources is to trust all of them:

```
grant
{
   permission javax.management.MBeanTrustPermission “register”;
};
```

The grant block below specifies a more restrictive policy that only trusts MBeans signed by *MyOrg*:

```
grant signedBy “MyOrg”
{
   permission javax.management.MBeanTrustPermission “register”;
};
```

Server Impact

The impact of this proposal on JMX client programs is minimal. If access is denied due to a lack of permission a `java.lang.SecurityException` is thrown. Since `SecurityException` is a subclass of `java.lang.RuntimeException` none of the MBeanServer methods that may throw it are obliged to declare it in their `throws` clause.

This proposal does impact MBeanServer implementations. Checks for each of the permissions specified above must be done at the appropriate points in the MBeanServer implementation code. How these checks are actually implemented is outside the scope of this proposal.

MBeanServerPermission and MBeanPermission checks are done in the usual way:

```
SecurityManager sm = System.getSecurityManager();
if (sm != null)
{
   sm.checkPermission(new <required Permssion>);
}
```

A more detailed note for implementations of MBeanServer.queryMBeans() and MBeanServer.queryNames().

First, clients that call these methods with no permissions will get back a SecurityException. If an MBeanPermission with *any* target name and an action name that contains “queryMBeans” or “queryNames” respectively is granted, then these methods will return a Set that is filtered upon the specified permissions: the Set will contain only the MBeans whose classnames and objectnames match those specified in the granted permissions, and no SecurityException will be thrown.

Second, clients that call these methods with a non-null QueryExp argument whose expression must access the MBeans to be applied (for example because contains a javax.management.AttributeValueExp object) must also have the MBeanPermission needed to apply the expression. If the client does not have the required MBeanPermission, then the MBeanServer implementation must not throw a SecurityException to the client, but instead must not return the MBean’s object name or object instance in the Set result of the MBeanServer.queryNames() or MBeanServer.queryMBeans() call, respectively.

Checking for MBeanTrustPermission requires a different approach. The check is done by determining whether or not the ProtectionDomain of the MBean to be registered implies the MBeanTrustPermission.