Getting Started with the Policy Manager

1. Obtaining and Installing the Software

The latest version of the Policy Manager software can be downloaded from the PERMIS website: http://sec.cs.kent.ac.uk/permis/ Installing it should be as simple as unzipping the zip file and executing the policyManager.jar file. The screen in Figure 1 should then be displayed:

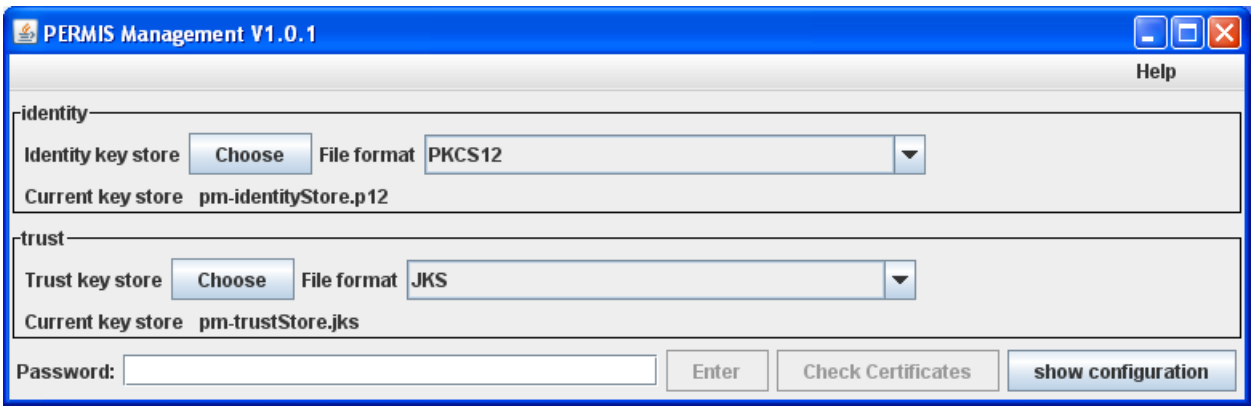

Figure 1. The initial configuration screen of Policy Manager

If you would like to try the Policy Manager in combination with the graphical Policy Tester, then you will also need to download the Policy Tester software which is available from the same location. The Policy Manager/Policy Tester combination is recommended for first time use, as it is the easiest way of seeing dynamic policy updates in action.

2. Configuration

A server/client architecture is used for dynamic policy updating, with the Policy Manager software playing the role of client and the managed PERMIS decision engine playing the role of server.

The Policy Manager software will connect to the server using an SSL connection, so key stores and certificates will be needed for strong authentication.

The Policy Manager doesn't need a lot of configuration apart from its key stores, as shown in Figure 1, and the location of the PERMIS engine, as shown in Figure 2. It's paramount that we have both client and server strong authentication, and that's exactly what these key stores provide. Both an identity key store and a trust store are needed.

2.1. The trust store

The trust store contains the public key certificates (PKCs) of trusted remote entities. On the client side (PM side) it should contain the PKC of the PERMIS engine. On the PERMIS engine side, it should contain

the PKC of the manager (Source of Authority) who writes and updates the PERMIS policy for the engine. The corresponding private keys are held in the identity key stores of their respective owners.

VERY IMPORTANT NOTE. A current limitation of the Java security software, is that if the PKC is not self signed, then the Java security software needs to have available to it the self signed certificate of the root CA that issued this PKC. But then anyone else who has a PKC from this root CA, whether it is in the trust store or not, is also trusted to make an SSL connection. Clearly this is not acceptable to the PERMIS engine, which must restrict its connection to the SOA only. This means that in the current release the manager (SOA) has to have a self signed certificate which is configured into the trust store of the PERMIS engine.

A client trust store for testing purposes is provided with the downloaded Policy Manager software, and this contains the PKC of the PERMIS engine configured into the Policy Tester software.

The filename of the test PM trust store is **pm-trustStore.jks**

The distinguished name of the test PERMIS engine in the self signed PKC is CN = Permis Engine, O = University of Kent, C = gb. However, the PKC in this store should **only be used for testing purposes** as the private key is shipped with the policy tester software. (Hence, its private key is not 'private' at all!)

2.2. The identity key store

The identity key store contains the *identity* of the owner of the key store. As such it contains the private key (and the public key certificate) of the owner. The trust store on the opposite side of the SSL connection has to contain the same PKC. In the Policy Manager software, the identity key store contains the private key (and the PKC) of the PERMIS policy writer (the SOA).

VERY IMPORTANT NOTE: the security of the installation is only guaranteed if the PKC in the trust store on the PERMIS engine side is a self signed one. Moreover, you shouldn't use the private key to sign any other PKCs. This way, you are assured that the entity mentioned in the PKC is the only one that can connect and update the policy.

An identity key store for testing purposes is shipped with the Policy Manager software.

The filename of the test PM identity key store is **pm-identityStore.p12**

Its private key should **only be used for testing purposes**. The distinguished name of the test manager is "CN = Arthur Smith, OU = Department of Chemistry, O = UCL, C = gb".

2.3 Password

At the moment, the implementation of the Policy Manager is such that the passwords to both the trust store and identity key store have to be identical.

The password for both the test stores is **password**

If you wish to do so you can check the contents of the PKCs in both key stores by pressing the Check Certificates button.

After you enter the password and press Enter, the screen in Figure 2 will be displayed.

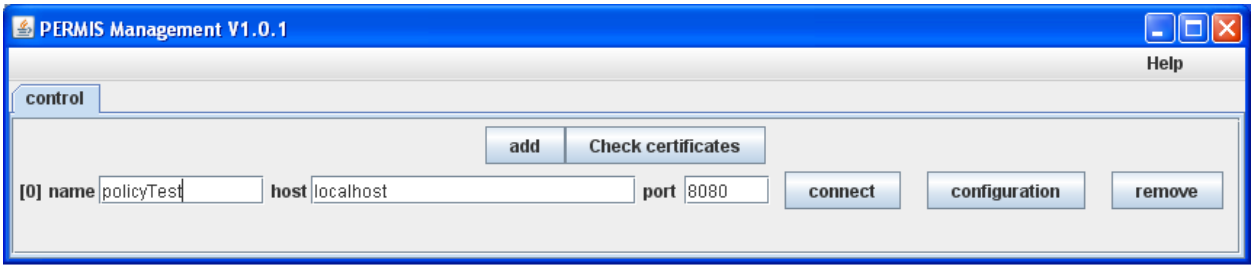

Figure 2. The configuration window for connecting to PERMIS engines.

2.4 Connecting to a PERMIS engine

You need to enter the DNS name and port number of the host on which the PERMIS engine is running, along with a name which will be used to name the tab of the window that is opened when the connection to PERMIS is opened (see Figure 3). When testing against a copy of Policy Tester running on the same machine as PM, you should enter localhost for the host name. Press “connect” and the window in Figure 3 will be displayed.

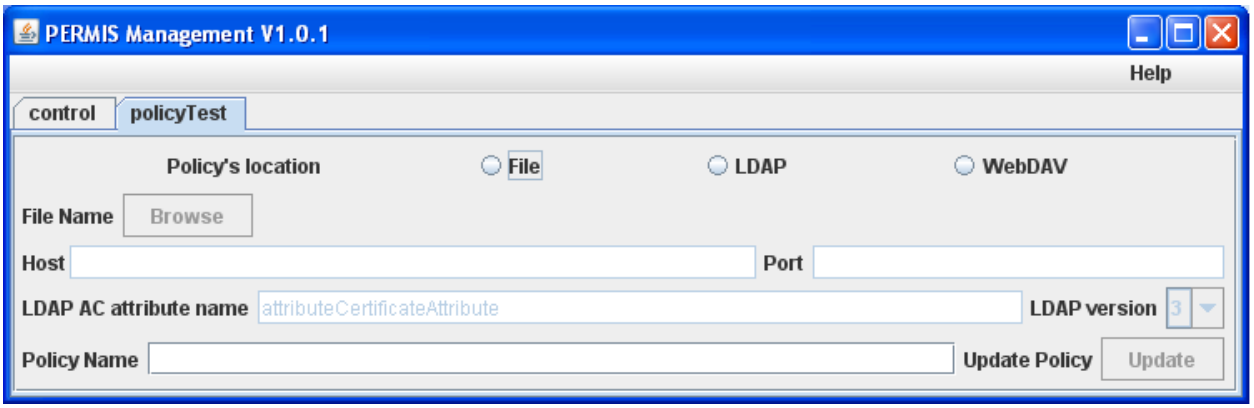

Figure 3. The Policy Manager Window

Full instructions on how to use the Policy Manager are given in Section 3.2. below.

3. Using Policy Manager with the Policy Tester

Here we will explain how to use the Policy Manager software together with the Policy Tester.

3.1 Preparing the Policy Tester

The Policy Tester software takes on the role of the server in this case. It will have to be configured correctly for use with the Policy Manager. You will need to have an identity key store and a trust store for the Policy Tester. Test versions of these stores are provided with the installation as follows:

Test Trust Store = trustSOAStore.jks

Test Identity Key Store = identityPERMISStore.p12

The identity key store has to contain the private key (and the PKC) of the PERMIS engine. An easy way to create this store is to use the openssl command line tool. (At the moment the policy tester software expects this store to be of type PKCS12.) You will also need a self signed certificate for the policy writer. The policy tester software expects this store to be of type JKS. Note: as explained above, it is very important that this trust store contains only the self signed PKC of the policy writer (as in the test trust store).

The locations of the stores can be set using Tools->configure and then choosing the Management configuration tab, as shown in Figure 4.

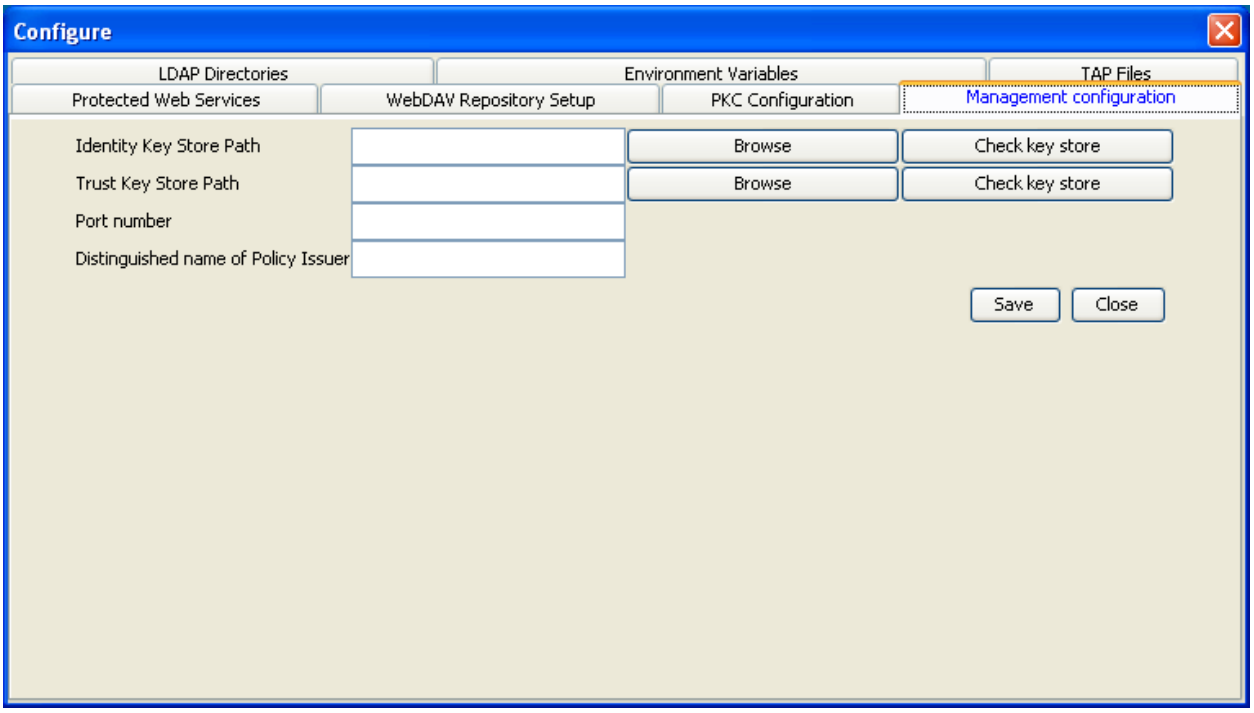

Figure 4. The Configuration Window from Policy Tester

You can then browse to the relevant stores. The default stores are “identityPERMISStore.p12” and “trustSOAStore.jks” and these are built into Policy Tester . The password of the stores that came with the software is simply 'password', and this is built into Policy Tester as well. If you use a different password, then you will be asked for it at the time when the software tries to read from the stores. This will happen the first time you click the Run Selected or Run All button (which are both part of the Run tab).

The Management configuration tab also give you the opportunity to check the content of PKCs in the key stores. This is done by clicking on the relevant Check key store button. In general this will ask you for the key store’s password.

In the Management configuration tab you can also specify the port number to use. The default port number is 8080.

Finally, the Management configuration tab allows you to give the name of the policy writer. This is only relevant when using text based policies (i.e. when using plain XML files) and hence is ignored when using signed policies. The name you give here has to match the name of the holder (and issuer) of the PKC in the trust store. In the test version the name should be "CN = Arthur Smith, OU = Department of Chemistry, O = UCL, C = gb".

Next you have to open or set up a test case. In the Run tab, make sure to check the Allow Dynamic Updates of Policy check box. After selecting the policy (either from a file or from an LDAP server), click either the Run Selected or Run All button. This is very important: you will not be able to update your policy if you have not done this. (The reason is that the permis decision engine only gets constructed when one of these buttons is clicked.)

3.2 Using the Policy Manager

Start the Policy Manager either by double clicking the .jar file or by issuing the command java –jar policyManager.jar . On the first screen (Figure 1) you have to specify the location of the identity key store and trust store. After you have typed the password to access the key stores, click Enter.

If the password was correct, you will now see the control panel (Figure 2), which has only one button on it. Click this Add button to add a connection. You now have the possibility to give your connection a name (this is for your convenience only and will be used as the tab name for this connection), along with the host name (or IP number) and the port number on which the *to-be-managed* PERMIS engine is running. In case the Policy Tester and the Policy Manager are running on the same computer you can use localhost as hostname. The default port number is 8080.

When you click the Connect button, if the connection is successful a new tab and window will be created (see Figure 3). This tab allows you to select the policy that is to replace the current one. You can either choose a new policy from the local file store, from an LDAP server or from a WebDAV server.

Important remark: if the policy you've loaded into the Policy Tester was contained in an attribute certificate (i.e. you have used a signed policy) then the policy that replaces the old one also has to be signed (by the same policy writer). If the policy you've used in the policy tester was a text based policy (i.e. a plain XML file) then the policy that is going to replace this one also has to be a text based policy. In short: you cannot mix and match signed and non-signed policies.

When you have given sufficient information to determine the new policy, the Update button will be enabled. Clicking it will cause the policy inside the PERMIS engine to be updated. This will be confirmed by a pop up window.

If you now return to the Policy Tester and click one of the Run buttons again, you should see the effects of using a different policy.

Remark: the text next to Select a Policy label will *not* be changed. Indeed, the policy tester itself is unaware that the policy has been changed behind its back!

3.3 A First Test

The easiest way to see dynamic updates in action is as follows:

1. Use the Policy Editor to create two policies protecting the same resource but giving different privileges to different roles. Save these two policies as text files on your local hard drive.

2. Start up the Policy Tester using a standard configuration file. (The easiest way to ensure this is to remove the pt.cfg file in the top level directory.) After the Policy Tester has been started choose Tools->Configure and then go to the Management configuration tab. Type the following in the field "Distinguished name of Policy Issuer": "CN = Arthur Smith, OU = Department of Chemistry, O = UCL, C = gb". There is no need to touch any of the other fields, although you can check the key stores by pressing the relevant buttons.

3. Set up a test case by creating subjects, actions and targets. In the Run Tab choose your first policy. (This step assumes that you already know how to set up a test case in the Policy Tester.)

4. Click on the Run All tab and see what decisions have been made.

5. Leave the Policy Tester running and fire up the Policy Manager using a default configuration. (The easiest way to achieve this is to delete the pmconfiguration.properties file if this is present in the top level directory of the installation.) On the screen as shown in Figure 1 type 'password' as the password and click Enter.

6. Add a connection as shown in Figure 2 and then click connect.

7. On the next screen select "File" as the policy location and browse to the second policy on your hard drive. After selecting it, click the Update button. You should now see a popup window confirming that the policy has been updated successfully.

8. Go back to the Policy Tester and click the Run All button once more. You should now see the effect of the policy update.

4. Using the API

4.1 Using Java Management Extensions

The Java Management Extensions technology was used to allow remote management of the PERMIS decision engine. As such, if you are familiar with this technology you can write your Policy Enforcement Point (PEP) using this technology, taking advantage of its authentication and authorization features.

A useful guide for getting started with the PERMIS API can be found on the PERMIS website. However, rather than using a PermisRBAC object, you will have to use a ManagedPermisRBAC object. This ManagedPermisRBAC object is an MBean. Furthermore, the PolicyFinder object used also has to implement the ManagedPolicyFinder interface. The API has the following ManagedPolicyFinders available: ManagedRepositoryACPolicyFinder and ManagedSimplePERMISPolicyFinder. If you would like to code the remote management yourself, the PAConfiguration object passed into the constructor of ManagedPermisRBAC has to be null.

So, the basic idea is this: if you want to make your engine available for remote updates of the policy you create a ManagedPermisRBAC object rather than a PermisRBAC object, but the PAConfiguration object has to be null. Next, you use JMX to make your engine available for remote management.

You will have to take suitable precautions to ensure the thread safety of your installation.

Using the PAConfiguration interface (recommended)

If you prefer not to use JMX directly you can use the PAConfiguration interface. Behind the scenes this still uses JMX using SSL as connection protocol. However, you will only be confronted with the configuration of the SSL layer.

Important Remark: at the moment, the code doesn't use explicit authorization as to who can issue an update. So, in order to be secure the trust store on the server side has to contain the self signed PKC of the policy writer only.

In case you're not familiar with the PERMIS API, it will be useful to read the Programmers Guide to the PERMIS API which can be found on the PERMIS website.

The easiest way to get started with the management API is to use the FilePAConfiguration class which implements the PAConfiguration interface based on the assumption that the key stores are file based and located on the local file system. That way, all you have to do is put the correct locations and passwords into this object and use it to create a ManagedPermisRBAC object.

As in the JMX case you will also have to use the ManagedPolicyFinder interface. Available implementations are: ManagedRepositoryACPolicyFinder and ManagedSimplePERMISPolicyFinder. The latter is for use with text based (non-signed) policies while the formed should be used with policies contained in X509 attribute certificates. Once you have decided on a policy format this same format will have to be used by all 'new' policies as well.

Another thing that might happen is that the getAuthzDecision method throws an OldCredentialsException. This happens when the Subject passed in wasn't created using the current policy. It's up to the PEP writer to decide how to react to this. The safest thing to do is probably to call getCreds again.

Using this approach, the installation should be thread safe thanks to the internal locking mechanism used.