Padlock 2.2 Integration Guide

Padlock 2.2 Integration Guide

The latest version of this document is always available on the Padlock Support Wiki at
http://support.padlocksoftware.net/display/padlock/Padlock+2.2+Integration+Guide

1) Using Padlock in an application

Requirements

Padlock is designed to be easily integrated into your application or library. As such, the only requirements for integration are Java 1.6+ (on either the Sun or IBM JVMs). To bundle Padlock simply add padlock-2.2.jar and the commons-codec-1.3.jar into your application's classpath.

2) Getting a license object

One of the first steps in validating a license is to get a reference to a Padlock license object. Padlock offers several methods, but most common is the use of license files. That's where this example will start. To import a license from a license file, simply use the static LicenseIO methods:

Code sample 1:

```
package net.padlocksoftware.padlockexamples;

import java.io.File;
import java.io.IOException;
import net.padlocksoftware.padlock.license.ImportException;
import net.padlocksoftware.padlock.license.License;
import net.padlocksoftware.padlock.license.LicenseIO;

public class SimpleImport {

  public static void main(String[] args) {

    try {
      License l = LicenseIO.importLicense(new File("license.lic"));
    } catch (IOException ex) {
      ex.printStackTrace();
      // Show an error messsage and leave software features disabled
} catch (ImportException ex) {
      ex.printStackTrace();
      // Show an error messsage and leave software features disabled
}
  }
}
```

The above code will obtain the license reference. Several issues can occur when importing licenses, such as the license file not existing or a malformed file. Simply catch these and optionally display an error to the user.

LicenseIO allows for other methods of creating license references, such as importing from a single String or any type of InputStream. These behave similarly to the above example and can easily be switched in.

3) Simple Validation

The basic validation code is quite simple, all that is required is a public key and the license to be validated. See the following example:

```
package net.padlocksoftware.padlockexamples;

import java.io.File;
import java.io.IOException;
import net.padlocksoftware.padlock.license.ImportException;
import net.padlocksoftware.padlock.license.License;
import net.padlocksoftware.padlock.license.LicenseIO;
import net.padlocksoftware.padlock.validator.Validator;
import net.padlocksoftware.padlock.validator.ValidatorException;

public class SimpleValidation {

  private static final String publicKey = "308201b83082012c06072a8648ce38040130" +
    "82011f02818100fd7f53811d75122952df4a9c2eece4e7f611b7523cef4400c31e3f" +
    "80b6512669455d402251fb593d8d58fabfc5f5ba30f6cb9b556cd7813b801d346ff2" +
    "6660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c6a8a6150f04" +
    "fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c702" +
    "15009760508f15230bccb292b982a2eb840bf0581cf502818100f7e1a085d69b3dde" +
    "cbbcab5c36b857b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe671" +
    "07108180b449167123e84c281613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1e" +
    "2bb3a675916ea37f0bfa213562f1fb627a01243bcca4f1bea8519089a883dfe15ae5" +
    "9f06928b665e807b552564014c3bfecf492a038185000281810 08aa4a3525415c2d0" +
    "42a320d605f190319ff1835805b287071a74b3b265e1bd316450c7f418fc6b1f0e94" +
    "d6c60a149555ae80e6e0bae72e9d8139ca72e8e6f108e75e5ca89613f78dbdd0d246" +
    "2a8256d383f0f3a60dc21903caa8742413e28335032f1e27e459e0272b12ceecc62c" +
    "d03a08f23b38d7212744c7addf4df084771a";

  public static void main(String\[\] args) {

    License l = null;
    try {
      l = LicenseIO.importLicense(new File("license.lic"));
     } catch (IOException ex) {
       ex.printStackTrace();
       // Show an error messsage and leave software features disabled
} catch (ImportException ex) {
       ex.printStackTrace();
       // Show an error messsage and leave software features disabled
}

     Validator v = new Validator(l, publicKey);
     LicenseState state;

     try {

       state = v.validate();
       // The validation was successful.  Enable full functionality

     } catch (ValidatorException ex) {

       // One or more tests failed.  Let's get the tests that
// tests that failed and print them out.
state = ex.getLicenseState();

       System.out.println("Failed Tests:");
       for (TestResult result : state.getFailedTests()) {
          System.out.println(result.getTest().getName() + " - " +
             result.getResultDescription());
       }

     }
  }
}
```

Notice the addition of the public key at the start as well as the Validation block at the end. The public key is just as it is named - public. It's ok for your users to know about it, and even see it. Do not, however, use it in a way that allows them to change it. In this case marking it as private and final does the job.

Note: To obtain the public key text, simply open up the KeyPair file you've created (either through the API, console tools, or the PadlockUI and copy the text shown for the PublicKey). They public key is also displayed at creation time when using the KeyMaker console tool, or can be viewed anytime in Padlock Manager's key's dialog.

The validation block at the end just creates the default Validator, using the public key and the license to be validated as constructor parameters. We then call validate(), which will determine the validity of the license. If the license is valid, nothing happens, all is peaceful and right with the world. If for any reason the license can't be validated (whether it's expired, missing, or has been tampered with) the Validator will throw a ValidatorException with an appropriate error message. For example, if you were to open license.lic in a text editor and change any value by a sigle digit, the Validator would throw an exception and the license would be considered invalid.

You can trust the validator to ensure that the license has not been modified in any way since it was created by you. It also guarantees that the license you're validating is a license created with your private key. As long as you keep the private key safe, nobody else will be able to create a license that passes the validator. This is the heart of what Padlock does, it let's you sleep at night.

4) Complex Validation - Plugins

When license validation requires logic that goes above and beyond the basics, this becomes a good candidate for a Validator Plugin. The ValidatorPlugin interface defines how you can easily add your own logic to the validation process and is designed to keep your validation logic separate from your application logic.

```
package net.padlocksoftware.padlock.validator;

import net.padlocksoftware.padlock.license.License;

public interface ValidatorPlugin {
  /**
   * This method performs a specific validation test on a license instance.
   *
   * @param license The license being validated.
   * @param validationParameters Informational parameters specific to the license being tested
   * and the Validator managing the validation.
   * @return A TestResult instance describing the test and the result of the test.
   */
  public TestResult validate(License license, ValidationParameters validationParameters);

  /**
   * The developer defined name for this plugin.
   */
  public String getName();

  /**
   * The developer defined description for this plugin.
   */
  public String getDescription();

}
```

The ValidatorPlugin interface itself is quite simple. Here's a real world example of an implementation: Padlocks' BlacklistPlugin. This plugin checks the license in question to see if it's part of the Validator's license blacklist, and throws an exception if a match is found. Note that this is code is still under development and the final shipping version may be different:

```
/**
 * Copyright (c) 2009 Padlock Software LLC.
 *
 * The contents of this file are private and contain confidential trade secrets.
 * Any viewing, distribution, or usage is strictly prohibited.
 */
package net.padlocksoftware.padlock.validator.plugins;

import java.util.Set;
import java.util.logging.Logger;
import net.padlocksoftware.padlock.license.License;
import net.padlocksoftware.padlock.license.LicenseTest;
import net.padlocksoftware.padlock.license.TestResult;
import net.padlocksoftware.padlock.validator.ValidationParameters;
import net.padlocksoftware.padlock.validator.ValidatorPlugin;

/**
 * The BlacklistPlugin compares a license signature with a known
 * blacklist and searches for matching signatures.  If a match is
 * found, the license is marked invalid.
 *
 * @since 2.0
 */
public final class BlacklistPlugin implements ValidatorPlugin {

  private static final String NAME = "Padlock Blacklist Plugin";

  private static final String DESCRIPTION = "Built in plugin that searches for licenses matching a" +
          "user defined blacklist";

  private final Logger logger = Logger.getLogger(getClass().getName());

  @Override
  public TestResult validate(License license, ValidationParameters validationParameters) {
    boolean passed = true;

    Set<String> blacklist = validationParameters.getBlacklist();

    String signature = license.getLicenseSignatureString();
    if (blacklist.contains(signature)) {
      logger.fine("Found blacklisted license: " + signature);
      passed = false;
    }

    return new TestResult(LicenseTest.BLACKLIST, passed);
  }

  @Override
  public String getName() {
    return NAME;
  }

  @Override
  public String getDescription() {
    return DESCRIPTION;
  }
}
```

The main method of interest is *validate(License, ValidationParameters)* which returns a LicenseStatus instance. If the plugin considers a license to be valid, it simply needs to return a TestResult with a passing state.

Possible uses of the ValidatorPlugin interface are an NTP plugin which could reference an external time source, or a concurrency plugin which would check for concurrent users for a given license.

To add a plugin to the validation routine, simply call

```
validator.addPlugin(plugin);
```

Plugins are not guaranteed to execute on the order they are added to the Validator.

It's worth noting that even after a plugin returns a failing result, the Validator will continue the validation process so that the results return a full set of passing and failing results. There is one exception to this rule: If a license is not signed or the signature is determined to be invalid, no plugins are run. This is a security measure to prevent plugin attacks with carefully crafted licenses.

4) Hardware Locking

Padlock comes with a pluggable provider API for locking a license to a particular set of MAC addresses. By default it provides compatibility with most common OSes, and will ignore MAC addresses commonly used by virtual machines.

By default this is enabled and requires no special setup other than the hardware locking features of the license. Several options are available for modifying the default provider:

Create a new default provider instance

This can be used only the set of blacklisted (aka VM) addresses need be viewed or modified. To enable this simply instantiate a new default provider and install it:

```
    DefaultMacAddressProvider newProvider = new DefaultMacAddressProvider();
    MacAddresses.setMacAddressProvider(newProvider);
```

With this, you can view and/or modify the MAC blacklist collection directly:

```
    Map<String, byte[]> vmAddresses = getVirtualAddressesMap();
```

Where the map key is a descriptor for the byte array (eg: "VmWare Client Range") and the byte array are hex values matching the leftmost bytes of the address range. See the JavaDocs in the Padlock distribution for more details.

Implement your own provider

Implementing your own API is as simple as implementing a 3 method interface:

```
public interface MacAddressProvider {

  /**
   * The meat of the interface, this method should search the local system
   * for all network interfaces and return an unordered Set of MAC address
   * Strings.
   * @return An unordered Set of MAC address Strings.
   */
  public Set<String> getSystemMacAddresses();

  /**
   * A human readable name for this provider.
   */
  public String getName();

  /**
   * A human readable version String for this provider.
   */
  public String getVersion();
}
```

Once implemented, install it as described above:

```
MacAddresses.setMacAddressProvider(new CustomMacProvider());
```

Conclusion

This completes the overview of the Padlock 2.0 integration guide. If you have any questions or issues, please email support@padlocksoftware.net.