JADE-PKI 1.0 Manual

Amadeusz Piotr Żołnowski

June 18, 2012

Contents

1 Introduction

The purpose of the *JADE-PKI* add-on is to introduce a public key infrastructure into *JADE.* The add-on provides security for two areas: agent messaging and agent mobility.

1.1 Agent messaging

Agent messaging is secured by the `PKIAgentMessagingService`. It provides methods of encryption, signing, decryption and verification of the ACL messages. An agent does not perform these operation itself, but gives it to the service. The agent only marks a message whether it should be signed and/or encrypted.

1.2 Agent mobility

When the agent moves between containers its code and data are exposed to malicious modification. The `PKIAgentMobilityService` provides integrity of code and data of the agent and certification of identity of the agent. It achieves that by signing the code and the data with the private key of the container and attaching the certificate into the field of the agent. For the agent to be secured that way it needs to implement the `Signable` interface or, which is recommended, extend one of the implementation: `SMobileAgent` or `SAgent`.

2 Installation

1. Assume that `$JADEHOME` points to the root directory of your JADE installation.
2. Unpack `pkiAddOn-1.0.zip` into `$JADEHOME/add-ons/pki`. JAR files are built already in `lib` subdirectory.
3. Your `$CLASSPATH` should look as follows:

```
CLASSPATH=$JADEHOME/add-ons/pki/lib/pki.jar:\
  $JADEHOME/add-ons/pki/lib/pki-examples.jar:\
  $JADEHOME/lib/jade.jar:\
  $JADEHOME/lib/commons-codec/commons-codec-1.3.jar
```

3 Configuration

Before *JADE-PKI* services can be used, they need to be set up.

3.1 Setting up PKI

1. The first step is to set up a public key infrastructure. Please follow `http://www.ibm.com/developerworks/java/library/j-certgen/` to set up certificates chain to your needs. The minimal setup is one root CA and a certificate for every container signed by the root CA. `CommonName` should be set conforming to the following scheme:

```
my-host.my-domain/my-platform/my-container
```

2. The certificates cannot be used directly. They need to be loaded into Java key stores. There's the `pkiimport` tool in `tools` subdirectory of

$JADEHOME/add-ons/pki. To pack all trusted CAs (necessarily including the root CA created at point 1) into the store, invoke pkiimport according to the following scheme:

```
./pkiimport truststore <keystore> <certfile1>:<alias1> \
    <certfile2>:<alias2> ...
```

where <keystore> is an output file, <certfileN> is a trusted CA and <aliasN> is an alias for that certificate by which the certificate can be referenced in a Java key store. You will be asked for password to set up for the key store.

3. The container key and corresponding certificate need to be put into another Java key store alone. For this pkiimport can be used, too. Invoke it according to the following scheme:

   ```
   ./pkiimport keystore <keystore> <keyfile> <certfile> \
       <alias>
   ```

   where <keystore> is an output file, <keyfile> is the private key of the container and <certfile> is its corresponding certificate. This step needs to be repeated for every container certificate created at the point 1.

The result of the steps above are the following files:

- truststore.jks – one Java key store containing trusted CAs,
- <container-name>-keystore.jks – a Java key store containing a private key and a certificate for a container named <container-name> – for each container.

3.2 Configuring JADE-PKI

Here goes the example file for the Main-Container:

```
# ---- JADE configuration ----
gui=true

# ------ Platform ------
name=myplatform
platform-id=myplatform
host=mymainhost
local-host=mymainhost
nomtp=true
```

```
# ------ Services  ------
services=\
jade.core.management.AgentManagementService;\
jade.core.messaging.MessagingService;\
jade.core.resource.ResourceManagementService;\
jade.core.mobility.AgentMobilityService;\
jade.core.event.NotificationService;\
jade.security.pki.core.PKICoreService;\
jade.security.pki.messaging.PKIAgentMessagingService;\
jade.security.pki.mobility.PKIAgentMobilityService

# ------ PKI configuration ------
jade_pki_keyStore=/path/to/maincontainer-keystore.jks
jade_pki_keyStorePassword=123456
jade_pki_trustStore=/path/to/truststore.jks
jade_pki_trustStorePassword=123456

# ---- end JADE configuration ----
```

Table 1: Parametrs for the `PKIAgentCoreService`.

Name	**Description**
`jade_pki_keyStore`	A path to a key store where a private key and a certificate of the container is in.
`jade_pki_keyStorePassword`	A password to the store pointed with the `jade_pki_keyStore` parameter.
`jade_pki_trustStore`	A path to a store where certificates of trusted CAs are in.
`jade_pki_trustStorePassword`	A password to the store pointed with the `jade_pki_trustStore` parameter.

The following classes are the *JADE-PKI* services:

- `jade.security.pki.core.PKICoreService`,
- `jade.security.pki.messaging.PKIAgentMessagingService`,
- `jade.security.pki.mobility.PKIAgentMobilityService`.

The first is a requirement for the remaining two. The second secures agent messaging and the last secures the agent when it moves between containers. Options in the `PKI configuration` section are used by the `PKICoreService` on initialization. They are described in the table 1.

And here goes an example of the configuration for an another container:

```
# ---- JADE configuration ----
container=true
container-name=Container-2

# ------ Platform ------
name=myplatform
platform-id=myplatform
host=mymainhost
local-host=anotherhost

# ------ Services  ------
services=\
jade.core.management.AgentManagementService;\
jade.core.messaging.MessagingService;\
jade.core.resource.ResourceManagementService;\
jade.core.mobility.AgentMobilityService;\
jade.core.event.NotificationService;\
jade.security.pki.core.PKICoreService;\
jade.security.pki.messaging.PKIAgentMessagingService;\
jade.security.pki.mobility.PKIAgentMobilityService

# ------ PKI configuration ------
jade_pki_keyStore=/path/to/anothercontainer-keystore.jks
jade_pki_keyStorePassword=987654
jade_pki_trustStore=/path/to/truststore.jks
jade_pki_trustStorePassword=123456

# ---- end JADE configuration ----
```

Another container has different key store with its private key and certificate, but uses a copy of the same file with the trusted CAs. Please take notice of the `Platform` section. The name, id and a host are the same as those of the main container, but `local-host` is the host name of this another container.

4 Basic usage

4.1 Usage with PKIAgentMessagingService

To make use of the message signing and encryption the agent needs to start `SendCertificateBehaviour` behaviour which is in `jade.security.pki.-messaging.behaviours` package. This can be done in `setup()` method of the agent as follows:

```
@Override
```

```
protected void setup() {
    addBehaviour(new SendCertificateBehaviour(this));
}
```

The message is encrypted, signed, decrypted and verified by the service. The agent only marks the message for the operation which can be done as follows:

```
/* Retrieve the service helper.  The ServiceException is
 * thrown if the service isn't initialized on the
 * current container, but for clarity it's not caught
 * here.  */
PKIAgentMessagingHelper pki = (PKIAgentMessagingHelper)
    getHelper(
    "jade.security.pki.messaging.PKIAgentMessaging");

/* Create a message.  A performative can be anything,
 * of course. */
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);

/* Set msg content, sender and so on... */

/* The message is marked for signing. */
pki.markForSignature(msg);

/* Before message is marked for encryption, an agent needs to
 * retrieve the receiver agent's container. */
if (pki.requestReceiversCertificate(myAgent, replyTo,
        1000)) {
    /* A certificate is retrieved, marking for encryption.  An
     * algorithm for a symmetric encryption is the AES in the
     * CBC mode and with the PKCS5 padding. */
    pki.markForEncryption(msg, "AES/CBC/PKCS5Padding",
            new KeySizeAlgorithmParameterSpec(128));
}

/* The message is send.  The JADE-PKI serivce handles it,
 * encrypts and signs (in that order). */
send(msg);
```

4.2 Usage with PKIAgentMobilityService

The agent to be secured with the service needs to implement the `Signable` interface. It is easier, however, to extend the `SMobileAgent` class which implements most of `Signable` methods.

```
public class SMobAgentExmpl extends SMobileAgent {

    private int constantValue;
    private int counter;

    /* This method needs to return data which needs to be
     * protected when agent moves between containers.  In
     * the example the counter field will be signed.
     * The data needs to be returned in the form of byte
     * array.  */
    @Override
    public byte[] getMutableData() {
        return ByteBuffer.allocate(4).putInt(counter).array();
    }

    /* This method needs to return data which is not going to
     * be changed after its initialization.  In the example the
     * constantValue field will be signed.  */
    @Override
    public byte[] getImmutableData() {
        return ByteBuffer.allocate(4).putInt(constantValue
            ).array();
    }

    @Override
    protected void setup() {
        counter = 0;
        constantValue = 3;

        /* This method request the service to sign the agent
         * class, identifier (AID) and data returned with
         * getImmutableData(). */
        ownMe();
    }

    @Override
    protected void afterMove() {
        /* We check if the move was successful.  If the agent
         * has been modified during transfer, then the
         * verification has failed and getSafeMoveStatus() will
         * return value different than SAFE_MOVE_OK.  */
        if (getSafeMoveStatus() != SAFE_MOVE_OK) {
            return;
        }
        /* On each container this value can be modified.  It
         * is signed when the agent leaves container.
         */
        counter++;
    }
}
```

4.3 Combining PKIAgentMessagingService and PKIAgent-MobilityService

To combine the use of both services it is best to extend the `SAgent` class which starts the `SendCertificateBehaviour` and extends `SMobileAgent`.