Table of Contents

A modern authentication system offers more than one check for user verification. A simple mechanism is username/password check. But the problem is often that it's not secure enough because if someone knows your username/password combination, doors are open. So an additional check is necessary to get better security. There are different techniques like:

- enter a verification code, sent to a custom email address
- use an app to confirm (often used for online banking)
- use another authentication provider like Auth0, Okta, Google, ...

Sure, the list is not complete but it covers common solutions.

The authentication mechanism of JVx is flexible. The default security managers are ready to use and offer base checks. The framework supports different databases and also xml files for user management. It's easy to create your own security manager if you need specific checks. If you require multi-factor authentication for your application. it's also possible with JVx because it's already built-in. JVx contains a standard implementation for common multi-factor authentication mechanism'. As usual, it's supported to implement your own multi-factor authentication.

The configuration is simple. Just wrap your existing security manager:

before:

```
<securitymanager>
  <class>com.sibvisions.rad.server.security.XmlSecurityManager</class>
  <userfile>users.xml</userfile>
</securitymanager>
```

after:

```
<securitymanager>
  <class>com.sibvisions.rad.server.security.mfa.MFASecurityManager</class>
  <mfa enabled="true">
    <class>com.sibvisions.rad.server.security.XmlSecurityManager</class>
    <authenticator>
<class>com.sibvisions.rad.server.security.mfa.auth.TextInputMFAuthenticator<
/class>
      <!--
        use an implementation of:
com.sibvisions.rad.server.security.mfa.auth.IPayloadNotificationHandler
        <notificationhandler></notificationhandler>
      -->
    </authenticator>
  </mfa>
  <userfile>users.xml</userfile>
</securitymanager>
```

The important thing is the authenticator. The `TextInputMFAuthenticator` creates a password/code for confirmation. It sends the code via `notificationhandler`. The default implementation sends an email. It's possible to create your own notification handler to send the code as SMS or use a different communication channel.

The default implementation is

`com.sibvisions.rad.server.security.mfa.auth.DefaultTextInputNotificationHandler`. It reads the mail server configuration from config.xml:

```
<mail>
  <smtp>
    <host>mail.server.com</host>
    <port>587</port>
    <username>user</username>
    <password>pwd</password>
    <tlsenabled>true</tlsenabled>
    <defaultsender>Noreply &lt;noreply@server.com&gt;</defaultsender>
  </smtp>
</mail>
```

The email is configured as template. The standard templates are located in package `/com/sibvisions/rad/server/security/mfa/auth/`. If you create custom templates, define your own package via `searchpath`:

```
<authenticator>
  <searchpath>com/myapp/mfa/auth</searchpath>
</authenticator>
```

The template mechanism loads different files. The first one is the translation file and the second one is the template html:

- package/translation_mfa_<language_code>.xml
- package/translation_mfa.xml (fallback: if no language specific xml file was found)
- package/confirmationcode_<language_code>.html
- package/confirmationcode.html (fallback: if no language specific html file was found)

The template should contain placeholders: [CONFIRMATION_CODE], [TIMEOUT].
This placehoders will be replaced with generated values.

If no template was found, a standard email with following text will be sent:

```
Confirmation code: [CONFIRMATION_CODE] is valid for [TIMEOUT] seconds
```

The UI will look like this screenshot:

The `TextInputMFAuthenticator` is one of three default authenticators. The other two are

- `com.sibvisions.rad.server.security.mfa.auth.WaitMFAuthenticator`
- `com.sibvisions.rad.server.security.mfa.auth.AbstractURLMFAuthenticator`

If you configure the `WaitMFAuthenticator`, the UI will look like this screenshot:

The wait authenticator can be use to wait for verification. The verification process is not included in JVx. You have to implement your own

`com.sibvisions.rad.server.security.mfa.auth.IWaitNotificationHandler` and add it to the configuration, e.g.:

```
<authenticator>
  ...
<notificationhandler>com.myapp.mfa.auth.MyWaitNotificationHandler</notificationhandler>
</authenticator>
```

The timeout for verification can be configured in config.xml as well, e.g.

```
<securitymanager>
  <mfa>
    <!-- milliseconds, 10 seconds -->
    <timeout>10000</timeout>
  </mfa>
</securitymanager>
```

If you configure a timeout which is lower 1, the default timeout of 5 minutes (300000) will be used.

The last built-in MF authenticator is the `AbstractURLMFAuthenticator`. This authentication method is not fully implemented because it requires an external service for user authentication. The URL authentication mechanism requires your own extension of `com.sibvisions.rad.server.security.mfa.auth.AbstractURLMFAuthenticator`. The class requires two methods:

```
/**
 * Creates a new {@link Link}.
 *
 * @param pToken the access token
 * @param pSession the session
 * @param pUser the user information
 * @return the URL
 */
protected abstract Link createLink(AccessToken pToken, ISession pSession,
UserInfo pUser);

/**
 * Gets whether the confirmation is finished.
 *
 * @param pToken the access token
 * @param pSession the session
 * @return <code>true</code> if confirmation is successful,
<code>false</code> otherwise
 */
protected abstract boolean isConfirmed(AccessToken pToken, ISession
pSession);
```

The implementation shouldn't be a problem. Simply send the link to the external authentication system and check if user is verified/authenticated. That's all.

The UI will show the URL as link or embedded in an iframe:

If default MFA implementations of JVx do not fit your needs, it's no problem to implement your own MF authentication. The MFA support is available in the MFAHandler class and this class can be used in your own security managers or MF authenticators.

```
<securitymanager>
  <class>com.sibvisions.rad.server.security.mfa.MFASecurityManager</class>
  <mfa enabled="true">
    <class>com.sibvisions.rad.server.security.XmlSecurityManager</class>
    <authenticator>
<class>com.sibvisions.rad.server.security.mfa.auth.MultiWaitMFAuthenticator<
/class>
    </authenticator>
  </mfa>
  <userfile>users.xml</userfile>
</securitymanager>
```

The full source code of the authenticator is available here.