

# Quarkus - Elytron Properties File Config

Quarkus provides support for properties file based authentication that is intended for development and testing purposes. It is not recommended that this be used in production as at present only plaintext and MD5 hashed passwords are used, and properties files are generally too limited to use in production.




Add the following to your `pom.xml`:








```
<dependencies>
  <!-- Elytron Security extension -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-elytron-security-properties-
file</artifactId>
  </dependency>
</dependencies>
```

## Configuration

The elytron-security-properties-file extension currently supports two different realms for the storage of authentication and authorization information. Both support storage of this information in properties files. The following sections detail the specific configuration properties.

 Configuration property fixed at build time -  Configuration property overridable at runtime

Property Files Realm Configuration	Type	Default
 <code>quarkus.security.users.file.realm-name</code> The realm name. This is used when generating a hashed password	string	Quarkus
 <code>quarkus.security.users.file.enabled</code> Determine whether security via the file realm is enabled.	boolean	false
 <code>quarkus.security.users.file.plain-text</code> If the properties are stored in plain text. If this is false (the default) then it is expected that the passwords are of the form HEX( MD5( username ":" realm ":" password ) )	boolean	false

 <code>quarkus.security.users.file.users</code>		
Classpath resource name of properties file containing user to password mappings. See <a href="#">Users.properties</a> .	string	<code>users.properties</code>
 <code>quarkus.security.users.file.roles</code>		
Classpath resource name of properties file containing user to role mappings. See <a href="#">Roles.properties</a> .	string	<code>roles.properties</code>
<b>Embedded Realm Configuration</b>	<b>Type</b>	<b>Default</b>
 <code>quarkus.security.users.embedded.realm-name</code>		
The realm name. This is used when generating a hashed password	string	<code>Quarkus</code>
 <code>quarkus.security.users.embedded.plain-text</code>		
If the properties are stored in plain text. If this is false (the default) then it is expected that the passwords are of the form HEX( MD5( username ":" realm ":" password ) )	boolean	<code>false</code>
 <code>quarkus.security.users.embedded.enabled</code>		
Determine whether security via the embedded realm is enabled.	boolean	<code>false</code>
 <code>quarkus.security.users.embedded.users</code>		
The realm users user1=password\nuser2=password2... mapping. See <a href="#">Embedded Users</a> .	<code>Map&lt;String, String&gt;</code>	<code>none</code>
 <code>quarkus.security.users.embedded.roles</code>		
The realm roles user1=role1,role2,...\nuser2=role1,role2,... mapping See <a href="#">Embedded Roles</a> .	<code>Map&lt;String, String&gt;</code>	<code>none</code>

## Properties Files Realm Configuration

The properties files realm supports mapping of users to password and users to roles with a combination of properties files. They are configured with properties starting with `quarkus.security.users.file`.

*example application.properties file section for property files realm*

```
quarkus.security.users.file.enabled=true
quarkus.security.users.file.users=test-users.properties
quarkus.security.users.file.roles=test-roles.properties
quarkus.security.users.file.realm-name=MyRealm
quarkus.security.users.file.plain-text=true
```

## Users.properties

The `quarkus.security.users.file.users` configuration property specifies a classpath resource which is a properties file with a user to password mapping, one per line. The following [example test-users.properties file](#) illustrates the format:

*example test-users.properties file*

```
scott=jb0ss ①
jdoe=p4ssw0rd ②
stuart=test
noadmin=n0Adm1n
```

- ① User `scott` has password defined as `jb0ss`
- ② User `jdoe` has password defined as `p4ssw0rd`

This file has the usernames and passwords stored in plain text, which is not recommended. If `plain-text` is set to `false` (or omitted) in the config then passwords must be stored in the form `MD5 ( username : realm : password )`. This can be generated for the first example above by running the command `echo -n scott:MyRealm:jb0ss | md5` from the command line.

## Roles.properties

*example test-roles.properties file*

```
scott=Admin,admin,Tester,user ①
jdoe=NoRolesUser ②
stuart=admin,user ③
noadmin=user
```

- ① User `scott` has been assigned the roles `Admin`, `admin`, `Tester` and `user`
- ② User `jdoe` has been assigned the role `NoRolesUser`
- ③ User `stuart` has been assigned the roles `admin` and `user`.

## Embedded Realm Configuration

The embedded realm also supports mapping of users to password and users to roles. It uses the main `application.properties` Quarkus configuration file to embed this information. They are

configured with properties starting with `quarkus.security.users.embedded`.

The following is an example `application.properties` file section illustrating the embedded realm configuration:

*example application.properties file section for embedded realm*

```
quarkus.security.users.embedded.enabled=true
quarkus.security.users.embedded.plain-text=true
quarkus.security.users.embedded.users.scott=jb0ss
quarkus.security.users.embedded.users.stuart=test
quarkus.security.users.embedded.users.jdoe=p4ssw0rd
quarkus.security.users.embedded.users.noadmin=n0Adm1n
quarkus.security.users.embedded.roles.scott=Admin,admin,Tester,user
quarkus.security.users.embedded.roles.stuart=admin,user
quarkus.security.users.embedded.roles.jdoe=NoRolesUser
quarkus.security.users.embedded.roles.noadmin=user
```

As with the first example this file has the usernames and passwords stored in plain text, which is not recommended. If `plain-text` is set to `false` (or omitted) in the config then passwords must be stored in the form `MD5 ( username : realm : password )`. This can be generated for the first example above by running the command `echo -n scott:MyRealm:jb0ss | md5` from the command line.

## Embedded Users

The user to password mappings are specified in the `application.properties` file by properties keys of the form `quarkus.security.users.embedded.users.<user>=<password>`. The following [Example Passwords](#) illustrates the syntax with 4 user to password mappings:

*Example Passwords*

```
quarkus.security.users.embedded.users.scott=jb0ss ①
quarkus.security.users.embedded.users.stuart=test ②
quarkus.security.users.embedded.users.jdoe=p4ssw0rd
quarkus.security.users.embedded.users.noadmin=n0Adm1n
```

① User `scott` has password `jb0ss`

② User `stuart` has password `test`

## Embedded Roles

The user to role mappings are specified in the `application.properties` file by properties keys of the form `quarkus.security.users.embedded.roles.<user>=role1[,role2[,role3[,...]]]`. The following [Example Roles](#) illustrates the syntax with 4 user to role mappings:

### Example Roles

```
quarkus.security.users.embedded.roles.scott=Admin,admin,Tester,user  
①  
quarkus.security.users.embedded.roles.stuart=admin,user ②  
quarkus.security.users.embedded.roles.jdoe=NoRolesUser  
quarkus.security.users.embedded.roles.noadmin=user
```

- ① User **scott** has roles **Admin**, **admin**, **Tester**, and **user**
- ② User **stuart** has roles **admin** and **user**