Part 5a: Additional Credential Security – Spring Data JPA + Jasypt

Author: Justin Rodenbostel

Posted In: Custom Development, Digital Transformation

In this short addendum to my earlier series on Spring Boot, I'll be covering a fairly trivial problem whose solution I had a somewhat hard time finding an example of. Like normal, I hope this will help someone out who is just getting started, and you can start at the beginning if you'd like.

On my current project, we're working with a security-focused consulting firm named Jemurai (http://jemurai.com). Before I go further I wanted to mention that I learned about this tool and many, many other things security-related from one of their people (@mkonda). A great learning experience, and a pleasure to work with. Check them out.

Back to my current project. We had a goal to encrypt sensitive data at rest in properties files. One of the things you'll notice if you've been following through Part 5 is that the database credentials are stored, in plain text, in properties files. In this installment, we'll be encrypting the password in that credential using Jasypt, an encryption tool for Java.

As usual, for this installment, I've created a copy of the code from Part 5 and created a new project called Part 5 With Jasypt. It's committed to Github, ready for cloning.

Download and Configure

Add the jasypt dependency to your build.gradle file:

/build.gradle:

```
dependencies {
    compile('org.springframework.boot:spring-boot-starter-web')
    compile('org.springframework.boot:spring-boot-starter-security
    compile('org.springframework.boot:spring-boot-starter-data-jpa
    compile('org.thymeleaf:thymeleaf-spring4:2.1.2.RELEASE')
    compile('org.jasypt:jasypt-spring31:1.9.2') //<-here it is.
    runtime('mysql:mysql-connector-java:5.1.6')

    testCompile('junit:junit')
}
```

Next, download the jasypt distribution from http://jaspyt.org: http://sourceforge.net/projects/jasypt/files/jasypt/jasypt%201.9.2/. Since we'll be encrypting the database credentials that are currently stored in a properties file, follow the instructions on the jasypt site outlining using their CLI tools (http://www.jasypt.org/encrypting-configuration.html) to encrypt your credential. I used the command below to perform this task. The task was run from the '/bin' directory in the jasypt distribution. The param named "input" should be string you wish to encrypt, and the password param is the decryption key used to decode your password as it's being ingested by Spring during app startup.

```
Justins-MacBook-Pro:bin justin$ ./encrypt.sh input="password" passw
```

...and the output should look something similar to this:

```
----ENVIRONMENT-----------------

Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM 24.5

----ARGUMENTS-------------------

input: password
password: testtest

----OUTPUT----------------------

xpPrNtXz+SQmTYB0WQrc+2T8ZTubofox
```

Updating Properties

If you read the jasypt manual, you've probably already updated your properties file with the newly encrypted value. In order to decrypt it, we need to give jasypt a directive – an indicator that is used by jasypt to determine which values need to be decrypted while they're being ingested. I've updated my application.properties file to include my newly encrypted password and the encryption directive.

```
spring.datasource.url=jdbc:mysql://localhost:3306/beyond-the-exampl
spring.datasource.username=root
spring.datasource.password=ENC(xpPrNtXz+SQmTYB0WQrc+2T8ZTubofox)
spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.jpa.hibernate.dialect= org.hibernate.dialect.MySQLInnoDBDial
```

```
spring.jpa.generate-ddl=false
```

Updating Spring Config

Last, you'll need to update your Spring config to use Jasypt utilities to decrypt the password while ingesting the necessary properties prior to constructing our datasource.

```
@Value("${spring.datasource.driverClassName}")
private String databaseDriverClassName;

@Value("${spring.datasource.url}")
private String datasourceUrl;

@Value("${spring.datasource.username}")
private String databaseUsername;

private String databasePassword;

@Bean
public DataSource datasource() throws IOException {
    org.apache.tomcat.jdbc.pool.DataSource ds = new org.apache.tom
    ds.setDriverClassName(databaseDriverClassName);
    ds.setUrl(datasourceUrl);
    ds.setUsername(databaseUsername);
    ds.setPassword(getSecurePassword());

    return ds;
}

private String getSecurePassword() throws IOException {
    StandardPBEStringEncryptor encryptor = new StandardPBEStringEr
    encryptor.setPassword(System.getProperty("blogpost.jasypt.key"
    Properties props = new EncryptableProperties(encryptor);
    props.load(this.getClass().getClassLoader().getResourceAsStrea
    return props.getProperty("datasource.password");
}
```

Here you can see I've changed the reference to my "databasePassword" property – it is no longer being populated by the @Value annotation. You can also see that I've replaced the value passed to the mutator of the password property of the datasource bean with a reference to a helper method that retrieves our password.

In that method ("getSecurePassword"), there are a few things going on. We're retrieving the password the encryptor will use to decrypt the value stored in our properties file. Note that it's the same value we used on the CLI during encoding. Also note that I'm assuming the value used for this is stored in a system property. Anywhere you can read it from will work, although environment variables/system properties seem to be the best place from what I've heard anecdotally and read in the Jasypt docs. There are some tricks to passing JVM options to the bootRun task. See here and here for examples. You can see we're using an extension of java.util.Properties called 'EncryptableProperties'. This is a Jasypt class that understands how to react when reading a

property value enclosed by the directive we talked about above. After that, we're using these properties like we would use any other java.util.Properties class.

Testing

To test our changes, simply execute the app using gradle bootRun. If we did it property (and remembered to specify our jasypt password somewhere), we should notice no user-facing changes – just the same app we had before, but now with encrypted properties. Note that I specified the value of my SystemProperty in the gradle script.

Conclusion

I hope you found this entry a useful continuation of the Jasypt and Spring.io documentation. Check back soon for another installment!

PREV

NEXT

Chicago Office
SPR Headquarters
Willis Tower

233 South Wacker Drive
Suite 3500
Chicago, Illinois 60606

Phone: 312-756-1760
Fax: 312-756-1751

WISCONSIN

Milwaukee Office

789 North Water Street
Suite 100
Milwaukee, Wisconsin 53202

Phone: 414-224-7964
Fax: 414-274-0120