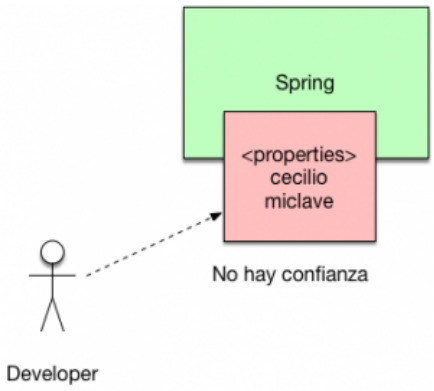
Uso de Spring Properties y encriptación

9 mayo, 2018 por Cecilio Álvarez Caules — 2 comentarios

El uso de **Spring Properties es muy común cuando trabajamos con Spring Framework**. Sin embargo hay situaciones en las cuales el manejo de propiedades **puede tener implicaciones no deseadas**. Uno de los casos más habituales es cuando en un fichero de propiedades se almacenan datos **"delicados"** como usuarios y contraseñas de una base de datos.



Spring y Confianza

No siempre la empresa tiene una confianza abs**oluta en los desarrolladores que contrata ya que en muchas casuísticas hay una cadena de subcontratas fuerte entre el desarrollador y el cliente final**. ¿Cómo podemos abordar esto?. Vamos a ver un ejemplo muy sencillo apoyándonos en Spring Framework y usando un hola mundo de Spring security. Para ello el primer paso es saber cuales son las dependencias de maven que vamos a utilizar:

```
    <dependencies>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>4.3.13.RELEASE</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-
```

```
            <version>4.3.13.RELEASE</version>
        </dependency>


        <!-- https://mvnrepository.com/artifact/org.springframework/spring-
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.3.13.RELEASE</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
        <dependency>
            <groupId>com.github.ulisesbocchio</groupId>
            <artifactId>jasypt-spring-boot</artifactId>
            <version>1.18</version>
        </dependency>


        <!-- https://mvnrepository.com/artifact/org.springframework.securit
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>4.2.2.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework.securit
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>4.2.3.RELEASE</version>
        </dependency>
</dependencies>
```

Spring Properties Configuración

El siguiente paso es configurar Spring a través del uso de anotaciones para que pueda arrancar, para ello vamos a usar un initializer y dos ficheros de configuración:

```
package com.arquitecturajava.init;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicati
import org.springframework.web.filter.DelegatingFilterProxy;
import org.springframework.web.servlet.DispatcherServlet;

import com.arquitecturajava.config.ConfiguracionSpring;

```

```
        ServletRegistration.Dynamic servlet = contenedor.addServlet("dispat
        servlet.setLoadOnStartup(1);
        servlet.addMapping("/");

        contenedor.addFilter("springSecurityFilterChain", new DelegatingFil
                .addMappingForUrlPatterns(null, false, "/*");
    }

}
```

Acabamos de inicializar el despachador de Spring y el filtro de seguridad, el siguiente paso es dar de alta los beans propios de Spring apoyandonos en anotaciones.

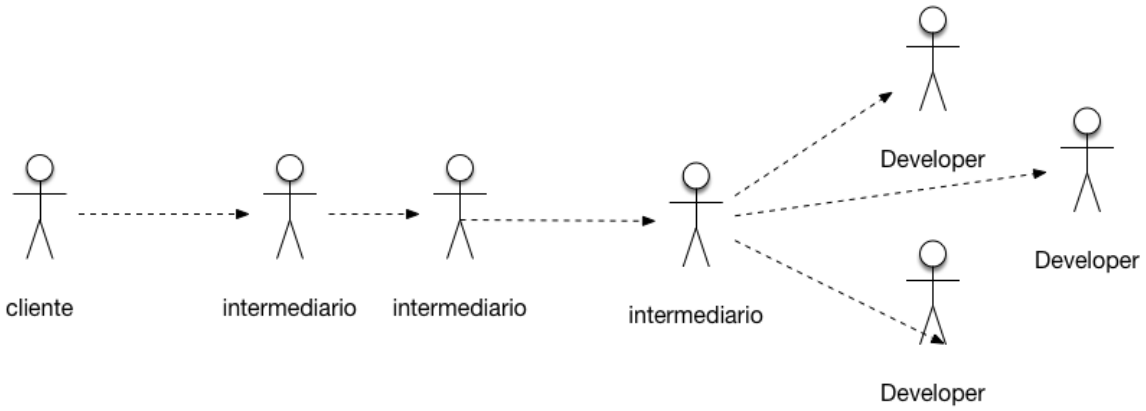
```
package com.arquitecturajava.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@ComponentScan("com.arquitecturajava.*")
@EnableWebMvc
@Import(ConfiguracionSeguridad.class)
public class ConfiguracionSpring {

    @Bean
    public ViewResolver internalResourceViewResolver() {
        InternalResourceViewResolver bean = new InternalResourceViewResolve
        bean.setViewClass(JstlView.class);
        bean.setPrefix("/WEB-INF/vistas/");
        bean.setSuffix(".jsp");
        return bean;
    }

}
```

```
package com.arquitecturajava.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builde
import org.springframework.security.config.annotation.web.builders.HttpSecu
import org.springframework.security.config.annotation.web.configuration.Ena
import org.springframework.security.config.annotation.web.configuration.Web

@Configuration
@EnableWebSecurity
public class ConfiguracionSeguridad  extends WebSecurityConfigurerAdapter{

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests().antMatchers("/**").hasRole("BASICO").and()
        super.configure(http);
    }

     @Autowired
        public void configureGlobal(AuthenticationManagerBuilder auth) thro
```

Por último un controlador para hacer pruebas:

```
1  package com.arquitecturajava.web;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  @Controller
6  public class HolaController {
7
8      @RequestMapping("/hola")
9      public String hola() {
10
11         return "hola";
12     }
13 }
```

En principio la aplicación es perfectamente válida sin embargo tenemos un problema. El usuario y la clave (“cecilio”, “miclave”) **son fácilmente visibles por parte de un desarrollador , los podemos externalizar a propiedades pero estaremos en la misma situación no conocemos al desarrollador.**



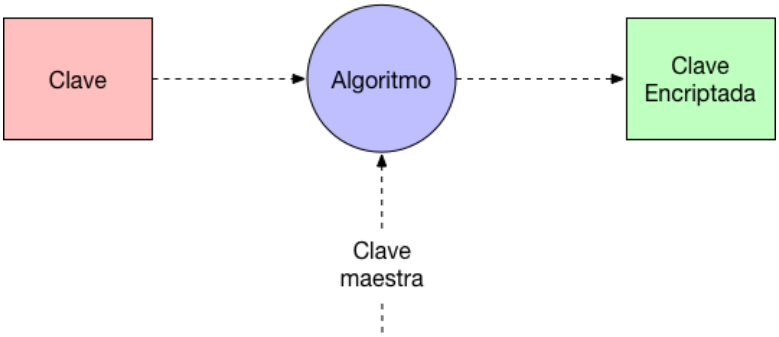
Spring y Criptografía

Esto en algunas estructuras de empresa genera problemas **y se necesita un paso de seguridad adicional**. Para ello vamos a apoyarnos en jasypt una librería criptográfica que se puede integrar de forma muy sencilla con spring y afectar a cualquier cadena que almacene el framework. Para ello **lo que utilizaremos es una clave maestra que se encarga de encriptar el resto de datos delicados.** Para poder realizar esta operación necesitaremos modificar el fichero de seguridad de Spring.

```
1  package com.arquitecturajava.config;
2
3  import java.io.IOException;
4  import java.util.Properties;
5
```

```java
@Configuration
@EnableWebSecurity
public class ConfiguracionSeguridad  extends WebSecurityConfigurerAdapter{


    //lo asigno con un menos -d a nivel de properties
    @Value("${clave_criptografica}")
    private String clave;
    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests().antMatchers("/**").hasRole("BASICO").and()
        super.configure(http);
    }

     @Autowired
        public void configureGlobal(AuthenticationManagerBuilder auth) thro
            auth
                .inMemoryAuthentication()
                    .withUser(getUsuarioSegura()).password(getPasswordSegur

        }


     private String getPasswordSegura() throws IOException {
            StandardPBEStringEncryptor encryptor = new StandardPBEStringEnc
            Properties props = new EncryptableProperties(encryptor);

            encryptor.setPassword(clave);
            props.load(this.getClass().getClassLoader().getResourceAsStream

            return props.getProperty("clave");
        }

     private String getUsuarioSegura() throws IOException {
            StandardPBEStringEncryptor encryptor = new StandardPBEStringEnc
            Properties props = new EncryptableProperties(encryptor);

            encryptor.setPassword(clave);
            props.load(this.getClass().getClassLoader().getResourceAsStream
            return props.getProperty("usuario");
        }

}
```

Como se puede observar hemos añadido un paso intermedio en el que se definen los métodos para leer una clave encriptada a través del fichero de propiedades. **¿Cómo funciona esto?** . Bueno necesitamos un algoritmo cryptográfico que se encargue de encriptar los datos utilizando una clave maestra.



Dcatalina.base="/Users/miusuario/GobiernoMayo/.metadata/.plugins/org.eclipse.wst.server.core/tmp0″ -Dcatalina.home="/Users/miusuario/Desktop/apache-tomcat-8.5.24″ -Dwtp.deploy="/Users/miusuario/GobiernoMayo/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps" -Djava.endorsed.dirs="/Users/miusuario/Desktop/apache-tomcat-8.5.24/endorsed" **–Dclave_criptografica=clavemaestra**

Generando las claves

El siguiente paso es encriptar el usuario y la clave utilizando nuestra "clavemaestra" . Para ello abrimos un terminal y ejecutamos sobre el jar de jasypt.

java -cp jasypt-1.9.2.jar org.jasypt.intf.cli.JasyptPBEStringEncryptionCLI input="cecilio" password="clavemaestra"

el resultado será:

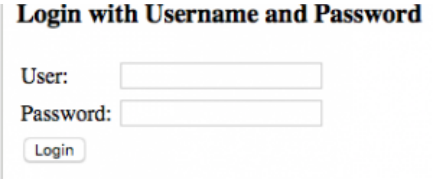
```
----ARGUMENTS-------------------

input: cecilio
password: clavemaestra



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

xEPg2YnZiInlKvnwYkx31g==
```

Ya tenemos encriptado el usuario nos falta encriptar su clave:

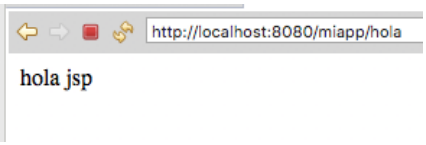
```
----ARGUMENTS-------------------

input: miclave
password: clavemaestra



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

AOAtq09c7ADqM6YfI4hNWw==
```

Es momento de ubicar ambos datos encriptados en nuestro fichero de propiedades que spring leerá.

usuario=ENC(xEPg2YnZiInlKvnwYkx31g==)



Introducimos “cecilio” de usuario y “miclave” como contraseña y accedemos:



Hemos accedido utilizando Spring Properties y su encriptación.

Otros artículos relacionados

1. Spring Security JDBC y su configuracion
2. Spring Security (I) configuracion
3. Spring PropertyPlaceHolderConfigurer

PDF

Archivada en: Spring Avanzado
Etiquetada con: SpringTips

Leave a Reply

2 Comments on "Uso de Spring Properties y encriptación"

Gabriel Espinel

Author

Cecilio Álvarez Caules

Gracias por el aporte . Es cierto que es mejorable 🙂

+ 0 − Reply

20 hours ago

BUSCAR

Buscar en este sitio ...

Mis Cursos de Java Gratuitos

Java Herencia

Java JDBC

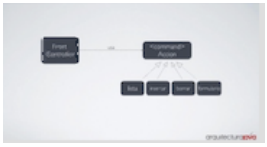
Servlets

Introduccion JPA

Mis Cursos de Java

Java Web

Pack Java Core



Arquitectura Java Solida con Spring

POPULAR

CONTACTO

contacto@arquitecturajava.com