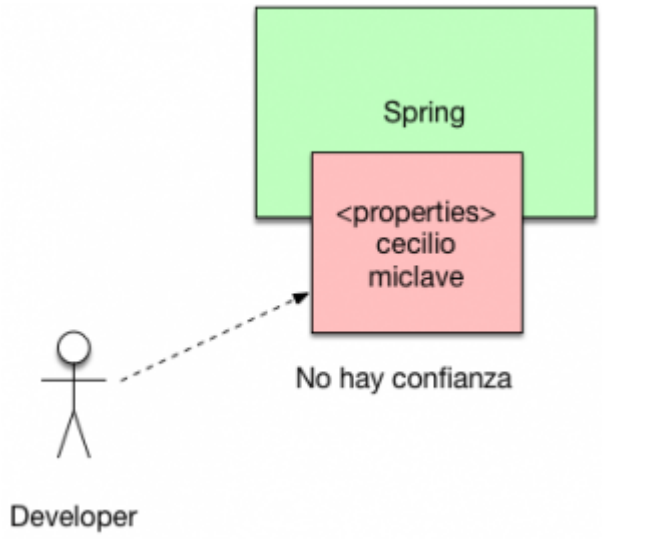
El uso de Spring Properties es muy común cuando trabajamos con Spring Framework. Sin embargo hay situaciones en las cuales el manejo de propiedades puede tener implicaciones no deseadas. Uno de los casos más habituales es cuando en un fichero de propiedades se almacenan datos “delicados” como usuarios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



Spring y Confianza

No siempre la empresa tiene una confianza absoluta 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-core -->
```

```
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>4.3.13.RELEASE</version>
		</dependency>

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

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


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

```
		</dependency>

		<!--
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<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.security/spring
-security-web -->
		<dependency>
<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-web</artifactId>
			<version>4.2.2.RELEASE</version>
		</dependency>
```

```
		<!--
https://mvnrepository.com/artifact/org.springframework.security/spring
-security-config -->
		<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.AnnotationConfigWebApplication
Context;
import org.springframework.web.filter.DelegatingFilterProxy;
import org.springframework.web.servlet.DispatcherServlet;
```

```
import com.arquitecturajava.config.ConfiguracionSpring;

public class SpringWebInitializer implements WebApplicationInitializer
{

        public void onStartup(ServletContext contenedor) throws
ServletException {

                AnnotationConfigWebApplicationContext contexto = new
AnnotationConfigWebApplicationContext();
                contexto.register(ConfiguracionSpring.class);
                // spring con la web
                contexto.setServletContext(contenedor);

                ServletRegistration.Dynamic servlet =
contenedor.addServlet("dispatcher", new DispatcherServlet(contexto));
                servlet.setLoadOnStartup(1);
                servlet.addMapping("/");

                contenedor.addFilter("springSecurityFilterChain", new
DelegatingFilterProxy("springSecurityFilterChain"))
                                        .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
InternalResourceViewResolver();
            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.builders
.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.Enabl
eWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSe
curityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class ConfiguracionSeguridad  extends
WebSecurityConfigurerAdapter{

        @Override
        protected void configure(HttpSecurity http) throws Exception {
http.authorizeRequests().antMatchers("/**").hasRole("BASICO").and().fo
rmLogin();
                super.configure(http);
        }
         @Autowired
            public void configureGlobal(AuthenticationManagerBuilder
auth) throws Exception {
                auth
                    .inMemoryAuthentication()
```

```
.withUser("cecilio").password("miclave").roles("BASICO");
            }
}
```

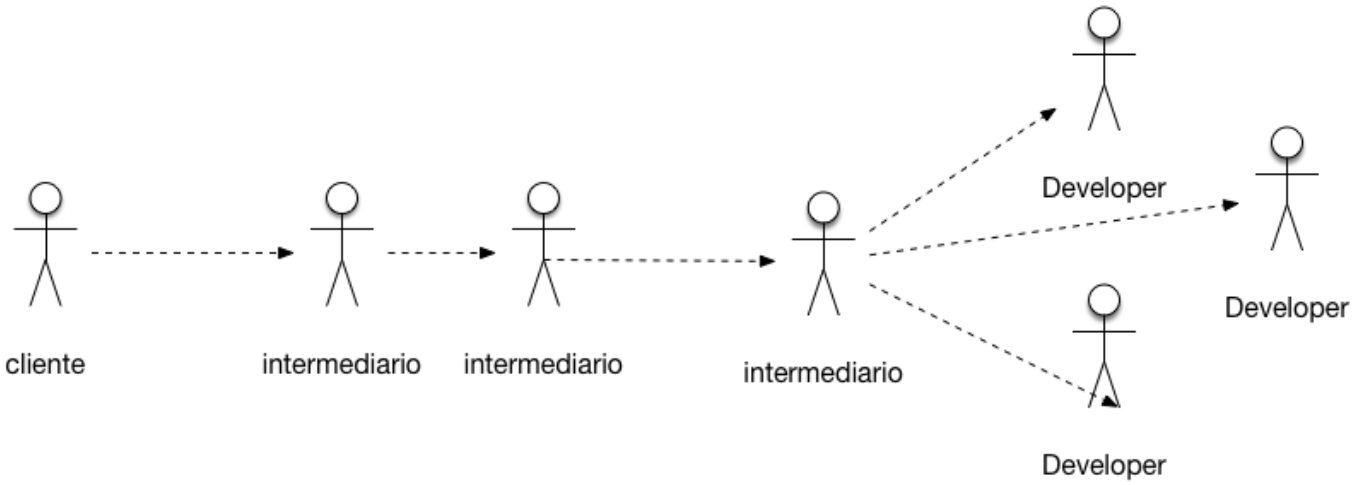
Por último un controlador para hacer pruebas:

```
package com.arquitecturajava.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HolaController {

        @RequestMapping("/hola")
        public String hola() {
                return "hola";
        }
}
```

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.

```
package com.arquitecturajava.config;

import java.io.IOException;
import java.util.Properties;

import org.jasypt.encryption.pbe.StandardPBEStringEncryptor;
```

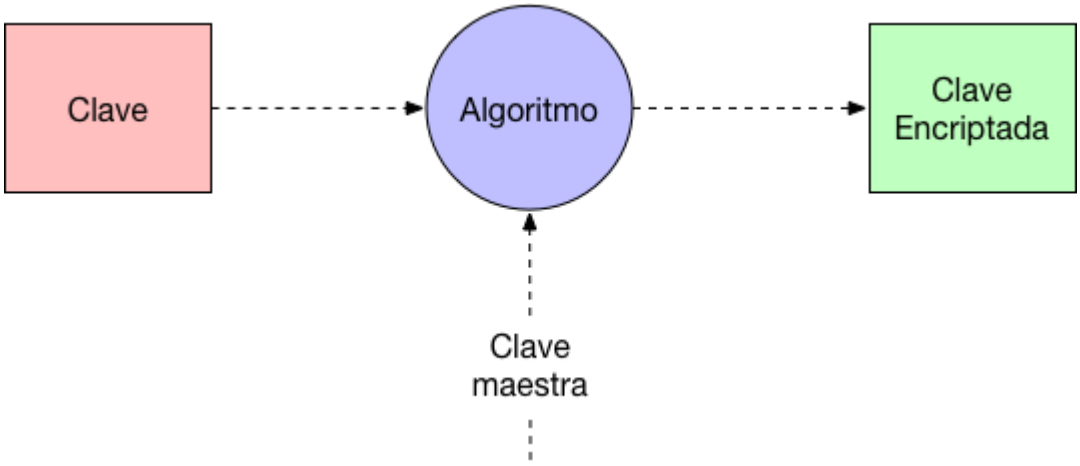
```
import org.jasypt.properties.EncryptableProperties;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders
.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.Enabl
eWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSe
curityConfigurerAdapter;

@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().fo
rmLogin();
                super.configure(http);
        }
         @Autowired
```

```
            public void configureGlobal(AuthenticationManagerBuilder
auth) throws Exception {
                auth
                    .inMemoryAuthentication()
.withUser(getUsuarioSegura()).password(getPasswordSegura()).roles("BAS
ICO");
            }
         private String getPasswordSegura() throws IOException {
                    StandardPBEStringEncryptor encryptor = new
StandardPBEStringEncryptor();
                    Properties props = new
EncryptableProperties(encryptor);
                    encryptor.setPassword(clave);
props.load(this.getClass().getClassLoader().getResourceAsStream("appli
cation.properties"));
                    return props.getProperty("clave");
                }
         private String getUsuarioSegura() throws IOException {
                    StandardPBEStringEncryptor encryptor = new
StandardPBEStringEncryptor();
                    Properties props = new
EncryptableProperties(encryptor);
                    encryptor.setPassword(clave);
props.load(this.getClass().getClassLoader().getResourceAsStream("appli
cation.properties"));
                    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.



Esta clave maestra la podemos pasar a través del concepto de propiedad.

```
//lo asigno con un menos -d a nivel de properties
@Value("${clave_criptografica}")
private String clave;
```

Lo más práctico para manejar esta clave maestra es que el administrador de sistemas la pase como una propiedad a la hora de arrancar el servidor Tomcat.

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:

Ya tenemos encriptado el usuario nos falta encriptar su clave:

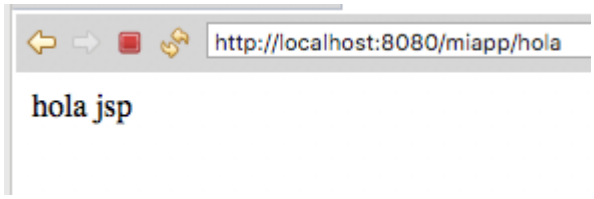
Es momento de ubicar ambos datos encriptados en nuestro fichero de propiedades que spring leerá.

usuario=ENC(xEPg2YnZiInlKvnwYkx31g==)

clave=ENC(AOAtq09c7ADqM6YfI4hNWw==)

Hemos terminado de configurar la aplicación con Spring Properties y encriptacion , la desplegamos en un tomcat y nos aparecerá el login por defecto de Spring Security.

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