TP3 - Sécuriser une application

Infrastructure de production

SÉCURISER UNE APPLICATION

Notation

Vous mettrez dans votre git tous les travaux que vous réaliserez, plusieurs seront choisis et noté. vous répondrez au questionnaire.

Préparation de lapplication :

Cette partie du TP est à faire dans l'application du TP précédent TP2 - spring cloud. Vous avez besoin d'avoir le module vets fonctionnel et actif dans consul.

- Configuration du SSL
- Configuration d'un provider username
- Configuration en http-basic
- Configuration en login-form

Utilisation du SSL

Création des certificats

1. Création du certificat principal

```
openssl genrsa -out ca/rootCA.key 2048
openssl req -x509 -new -nodes -key ca/rootCA.key -sha256 -days 1024 -out ca/rootCA.pem

>>>
-----
Country Name (2 letter code) [FR]:FR
State or Province Name (full name) [Loiret]:Loiret
Locality Name (eg, city) [Orleans]:Orleans
Organization Name (eg, company) [diogene Ltd]: Infra Ltd
Organizational Unit Name (eg, section) []:Infra Ltd Certificate Authority
Common Name (e.g. server FQDN or YOUR name) []:Infra Ltd Root CA
Email Address []:
```

2. Création du certificat de consul

```
openssl genrsa -out ca/consul.key 2048
openssl req -new -key ca/consul.key -out ca/consul.csr

>>>
-----
Country Name (2 letter code) [FR]:FR
State or Province Name (full name) [Loiret]:Loiret
Locality Name (eg, city) [Orleans]:Orleans
Organization Name (eg, company) [diogene Ltd]:Infra Ltd
Organizational Unit Name (eg, section) []:Infra Ltd Certificate Authority
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Signature du certificat par l'autorité principale

```
openssl x509 -req -in ca/consul.csr -CA ca/rootCA.pem -CAkey ca/rootCA.key -CAcreateserial -out ca
/consul.pem -days 500 -sha256
```

4. Création du certificat spring

```
openssl genrsa -out ca/spring.key 2048
openssl req -new -key ca/spring.key -out ca/spring.csr

>>>
-----
Country Name (2 letter code) [FR]:FR
State or Province Name (full name) [Loiret]:Loiret
Locality Name (eg, city) [Orleans]:Orleans
Organization Name (eg, company) [diogene Ltd]:Infra Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) [Le nom de votre machine en deptinfo.univ-orleans.fr]:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

5. Signature du certificat par l'autorité principale

```
openssl x509 -req -in ca/spring.csr -CA ca/rootCA.pem -CAkey ca/rootCA.key -CAcreateserial -out ca
/spring.pem -days 500 -sha256
```

6. Création du truststore

```
keytool -import -alias rootca -file  ca/rootCA.pem -keystore  ca/truststore.jks -storepass changeit
```

7. Création du certificat pour les applications spring

```
openssl pkcs12 -export -in ca/spring.pem -inkey ca/spring.key -chain -CAfile  ca/rootCA.pem -out ca
/server.p12 -name tomcat  -passout pass:Pa55w0rd
```

Démarrage des application

1. Démarrer consul

```
./consul.exe agent -dev -config-file ./config.json
```

2. Le fichier de configuration est

config.json

```
{
        "datacenter": "dc1",
        "log_level": "DEBUG",
        "data_dir": "P:\\dev\\consul.io\\data_dir",
        "ca_file": "P:\\dev\\ca\\rootCA.pem",
        "cert_file": "P:\\dev\\ca\\consul.pem",
        "key_file": "P:\\dev\\ca\\consul.key",
        "verify_incoming": true,
        "verify_outgoing": false,
        "server": true,
        "bootstrap": true,
        "verify_server_hostname": false,
        "ui":true
}
```

3. Préparer l'application **spring-petclinic-vets-service**. Cette application doit fonctionner dans consul et être visible de la partie admin
4. Utiliser les certificats générés précédemment

application.properties

```
management.security.enabled=false

security.require-ssl=true
server.ssl.enabled=true

key-store=classpath:server.p12
key-store-type=PKCS12
key-store-password=Pa55w0rd
key-alias=tomcat

trust-store=classpath:truststore.jks
trust-store-type=JKS
trust-store-password=changeit
```

5. Consulter le service https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot]/api/vets. Le navigateur présente une erreur lié a l'expiration et au faite que le certificat est autosigné. Vous pouvez visualiser les certificats grâce à keystore-explorer.
6. Le ssl étant mis. Consul montre des erreurs sur ce composant car l'appel est fait en http. La clé pour gérer le bon appel :

```
spring.cloud.consul.discovery.scheme: https
```

Un autre problème subsiste. Comme le certificat est autosigné, consul ne l'accepte pas. Ajouter le support de ce certificat est relativement long

7. Tenter un démarrage de votre code avec

```
java -Djavax.net.ssl.trustStore=/home/etud/diogene.moulron/TP/03/ca/truststore.jks -Djavax.net.ssl.
trustStorePassword=changeit -jar target/spring-petclinic-vets-service-2.0.4.jar
```

8. **Maintenant** réaliser un programme java lisant le service (https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot]/api/vets) de cette application en utilisant **org.apache.httpcomponents:fluent-hc:4.5.3,** quel est le message retourné ? Comment le corriger ? une piste : javax.net.ssl.trustStore
9. Tenter un démarrage de votre code avec

```
-Djavax.net.ssl.trustStore=/home/etud/diogene.moulron/TP/03/ca/truststore.jks -Djavax.net.ssl.
trustStorePassword=changeit
```

Utilisation de spring-security dans l'application :

Pour springifiy une application en ajoutant la sécurité

1. Supprimer le http pour le reste du TP cela permet de valider les différentes options dans consul
2. Consulter le service https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot]/actuator/health
3. Mettre dans l'application.properties la propriété permettant d'avoir tous les onglets.

   applicaiton.properties

   ```
   management.endpoints.web.exposure.include=*
   management.endpoints.jmx.exposure.include=*
   management.security.enabled=false
   ```

   Cela permet d'avoir accès a toutes les resources actuator. Hors ces ressources contiennent des données sensible qui peuvent aider a découvrir l'infrastructure.
4. ajout des informations dans le pom maven

   ```
   <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
   <dependency>
       <groupId>org.springframework.security</groupId>
       <artifactId>spring-security-test</artifactId>
       <scope>test</scope>
   </dependency>

   <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>javax.servlet-api</artifactId>
   </dependency>
   ```

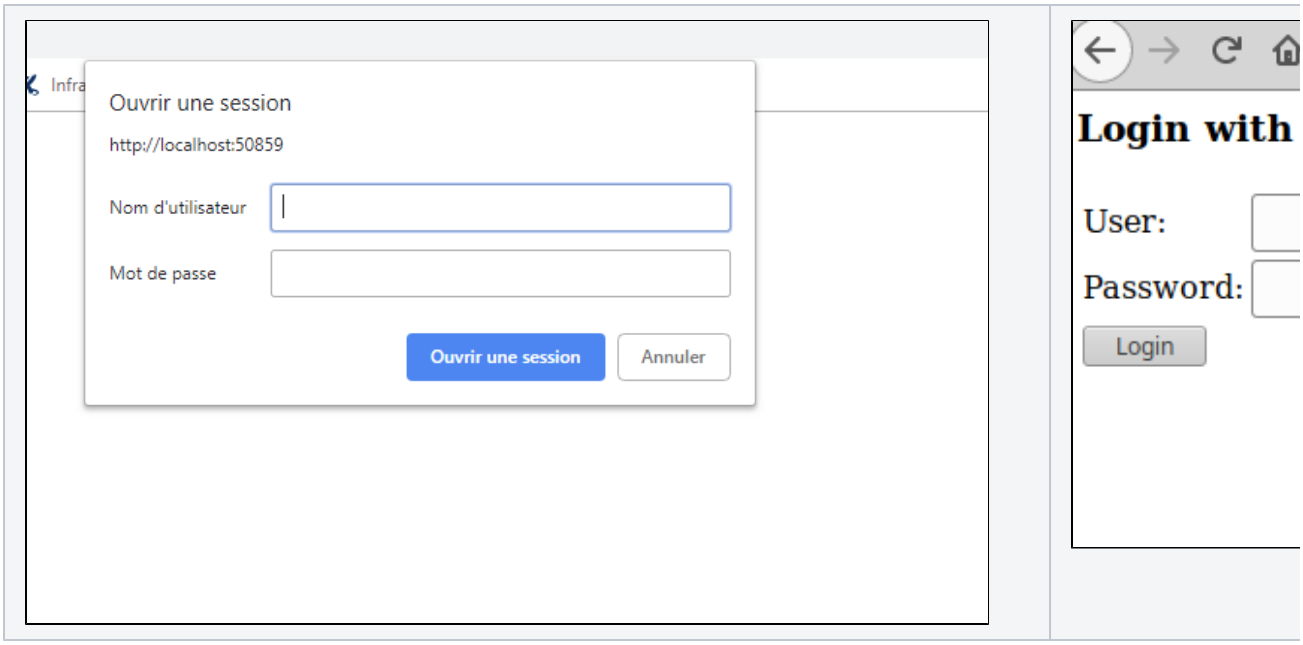
5. Activer la sécurité dans l'application

   ```
   import org.springframework.boot.SpringApplication;
   import org.springframework.boot.autoconfigure.SpringBootApplication;
   import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

   @SpringBootApplication
   @EnableWebSecurity
   public class VetsServiceApplication {

   }
   ```

6. Modification du test unitaire pour faire passer la securité (le mettre en ignore)
7. A partir de la votre application est sécurisé



Les informations d'authentification sont dans la console
Cette authentification est pour toutes l'application même le service de check health utilisé par consul. il faut gérer plus finement le mode d'authentification.

configurer plus finement le niveau de sécurité

1. créer une classe de configuration (spring-petclinic-vets-service\src\main\java\org\springframework\samples\petclinic\vets\config\SecurityConfig.java)
   exclure le démarrage automatique de certaines configurations

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.actuate.autoconfigure.security.reactive.
ReactiveManagementWebSecurityAutoConfiguration;
import org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration;
import org.springframework.boot.autoconfigure.security.servlet.UserDetailsServiceAutoConfiguration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@SpringBootApplication (exclude = {SecurityAutoConfiguration.class, UserDetailsServiceAutoConfiguration.
class,
        ReactiveManagementWebSecurityAutoConfiguration.class})
@EnableWebSecurity
public class VetsServiceApplication {

}
```

   Création d'une classe de configuration de la securité

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter { }
```

2. Ajouter une définition des utilisateurs. Il faut surcharger la bonne méthode de la classe de sécurité

```
auth.inMemoryAuthentication()
.withUser(User.withDefaultPasswordEncoder().username("admin").password("demo").roles("ADMIN"))
.withUser(User.withDefaultPasswordEncoder().username("user").password("demo").roles("USER"));
```

3. configurer la sécurité. Il faut surcharger la bonne méthode de la classe de sécurité

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

        http.csrf().disable()
                .authorizeRequests()
                    .anyRequest().authenticated()
                .and()
                .httpBasic()
                .and()
                .logout()
                    .permitAll();
                 //   .and()
                //.exceptionHandling().accessDeniedHandler(accessDeniedHandler);
    }
```

4. teste les services
    a. https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot] /actuator/health
    b. https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot] /vets
    c. https://[xxxx.deptinfo.univ-orleans.fr]:[votre prot] /actuator/metrics
5. Mettre un formulaire de login dans votre application

Le questionnaire

Le questionnaire est ici