Two way authentication in SpringBoot:

# Certification Generation

* Login to the linux servers (EC2) and execute below commands (openssl is working)

|  |
| --- |
| #Create folders to generate all files (separated for client and server)  mkdir ssl  cd ssl  mkdir client  mkdir server  ## Generate server private key and self-signed certificate in one step  openssl req -x509 -newkey rsa:4096 -keyout server/serverPrivateKey.pem -out server/server.crt -days 3650 -nodes  ## Create PKCS12 keystore containing private key and related self-sign certificate  openssl pkcs12 -export -out server/keyStore.p12 -inkey server/serverPrivateKey.pem -in server/server.crt  ## Generate server trust store from server certificate  keytool -import -trustcacerts -alias root -file server/server.crt -keystore server/trustStore.jks  ## Generate client's private key and a certificate signing request (CSR)  openssl req -new -newkey rsa:4096 -out client/request.csr -keyout client/myPrivateKey.pem -nodes  ## Server  # Sign client's CSR with server private key and a related certificate  openssl x509 -req -days 360 -in client/request.csr -CA server/server.crt -CAkey server/serverPrivateKey.pem -CAcreateserial -out client/pavel.crt -sha256  # Verify client's certificate  openssl x509 -text -noout -in client/pavel.crt  # Create PKCS12 keystore containing client's private key and related self-sign certificate  openssl pkcs12 -export -out client/client\_pavel.p12 -inkey client/myPrivateKey.pem -in client/pavel.crt -certfile |

# Server Configuration (Springboot)

Add following into the “application.properties”

|  |
| --- |
| *##Nitesh SSL conf  #server side (This will require client to send server’s CA certificate)* server.port=8443 server.ssl.key-store-type=PKCS12 server.ssl.key-store=classpath:keyStore.p12 server.ssl.key-store-password=changeit  *#Mutual Authentication( This will require client to present it’s certification as well)* server.ssl.trust-store=classpath:trustStore.jks server.ssl.trust-store-password=changeit server.ssl.trust-store-type=JKS server.ssl.client-auth=need |

# Sample Client Code

Maven Depedency:

|  |
| --- |
| <dependencies> <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>4.5.12</version> </dependency>  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpcore</artifactId>  <version>4.4.13</version>  </dependency> </dependencies> |

ConnectIT.java (Mutual Auth disabled on the server)

Note: Client just require server’s CA cert

|  |
| --- |
| import org.apache.http.HttpEntity; import org.apache.http.client.methods.CloseableHttpResponse; import org.apache.http.client.methods.HttpGet; import org.apache.http.config.Registry; import org.apache.http.config.RegistryBuilder; import org.apache.http.conn.HttpClientConnectionManager; import org.apache.http.conn.socket.ConnectionSocketFactory; import org.apache.http.conn.socket.PlainConnectionSocketFactory; import org.apache.http.conn.ssl.SSLConnectionSocketFactory; import org.apache.http.conn.ssl.SSLContexts; import org.apache.http.impl.client.CloseableHttpClient; import org.apache.http.impl.client.HttpClientBuilder; import org.apache.http.impl.conn.BasicHttpClientConnectionManager; import org.apache.http.util.EntityUtils;  import javax.net.ssl.SSLContext; import java.io.FileInputStream; import java.io.IOException; import java.io.InputStream; import java.security.\*; import java.security.cert.CertificateException;  */\*\*  \* Demonstrate connecting to a server secured with client-side SSL certificates.  \*/* public class ConnectIT {    */\*\*  \* PKCS12 file passphrase.  \*/* final String KEY\_STORE\_PASSWORD = "changeit";   */\*\*  \* URL to connect to. That is, a server for which the above certificate is required.  \*/* final String URL = "https://localhost:8443/apn/swagger-ui.html#/";   public void sslConnect() throws KeyStoreException, IOException, CertificateException,  NoSuchAlgorithmException, KeyManagementException, UnrecoverableKeyException {  System.*setProperty*("javax.net.ssl.trustStore", "C:\\Nitesh-SENA\\ANTrackerWeb\\APN\\doc\\SSL\\server\\trustStore.jks");  */\* Nitesh: Required if client auth is required at the server side   // \* Path to your client-side SSL certificate in the PKCS12 format, as generated by OpenSSL.  final String KEY\_STORE\_PATH = "C:\\Nitesh-SENA\\ANTrackerWeb\\APN\\doc\\SSL\\client\\client\_pavel.p12";   // Load the key store, containing the client-side certificate.  KeyStore keyStore = KeyStore.getInstance("pkcs12");  InputStream keyStoreInput = new FileInputStream(KEY\_STORE\_PATH);  keyStore.load(keyStoreInput, KEY\_STORE\_PASSWORD.toCharArray());  System.out.println("Key store has " + keyStore.size() + " keys");   // Create an SSL context with our private key store.  // We are only loading the key-material here, but if your server uses a self-signed certificate,  // you will need to load the trust-material (a JKS key-store containing the server's public SSL  // certificate) as well.   SSLContext sslContext = SSLContexts.custom()  .loadKeyMaterial(keyStore, KEY\_STORE\_PASSWORD.toCharArray())  .useTLS()  .build();  \*/* SSLContext sslContext = SSLContext.*getDefault*();   *// Prepare the HTTPClient.* HttpClientBuilder builder = HttpClientBuilder.*create*();  SSLConnectionSocketFactory sslConnectionFactory = new SSLConnectionSocketFactory(  sslContext, SSLConnectionSocketFactory.*ALLOW\_ALL\_HOSTNAME\_VERIFIER*);  builder.setSSLSocketFactory(sslConnectionFactory);  Registry<ConnectionSocketFactory> registry = RegistryBuilder.<ConnectionSocketFactory>*create*()  .register("https", sslConnectionFactory)  .register("http", new PlainConnectionSocketFactory())  .build();  HttpClientConnectionManager ccm = new BasicHttpClientConnectionManager(registry);  builder.setConnectionManager(ccm);   *// Perform a sample HTTP request.* try {  CloseableHttpClient httpClient = builder.build();  HttpGet httpget = new HttpGet(URL);  try {  CloseableHttpResponse response = httpClient.execute(httpget);  HttpEntity entity = response.getEntity();   System.*out*.println("----------------------------------------");  System.*out*.println(response.getStatusLine());  if (entity != null) {  System.*out*.println("Response content length: " + entity.getContentLength());  System.*out*.printf(EntityUtils.*toString*(entity));  }  EntityUtils.*consume*(entity);  } catch(Exception e) {  e.printStackTrace();  }  } catch(Exception e) {  e.printStackTrace();  }  }   public static void main(String[] args) throws Exception {  new ConnectIT().sslConnect();  } } |

ConnectIT.java (Mutual Auth enabled and server running on SSL)

Note: Client require server’s CA cert because of self-signed cert + client certificate (which is signed by the server)

|  |
| --- |
| import org.apache.http.HttpEntity; import org.apache.http.client.methods.CloseableHttpResponse; import org.apache.http.client.methods.HttpGet; import org.apache.http.config.Registry; import org.apache.http.config.RegistryBuilder; import org.apache.http.conn.HttpClientConnectionManager; import org.apache.http.conn.socket.ConnectionSocketFactory; import org.apache.http.conn.socket.PlainConnectionSocketFactory; import org.apache.http.conn.ssl.SSLConnectionSocketFactory; import org.apache.http.conn.ssl.SSLContexts; import org.apache.http.impl.client.CloseableHttpClient; import org.apache.http.impl.client.HttpClientBuilder; import org.apache.http.impl.conn.BasicHttpClientConnectionManager; import org.apache.http.util.EntityUtils;  import javax.net.ssl.SSLContext; import java.io.FileInputStream; import java.io.IOException; import java.io.InputStream; import java.security.\*; import java.security.cert.CertificateException;  */\*\*  \* Demonstrate connecting to a server secured with client-side SSL certificates.  \*/* public class ConnectIT {   */\*\*  \* Path to your client-side SSL certificate in the PKCS12 format, as generated by OpenSSL.  \*/* final String KEY\_STORE\_PATH = "C:\\Nitesh-SENA\\ANTrackerWeb\\APN\\doc\\SSL\\client\\client\_pavel.p12";   */\*\*  \* PKCS12 file passphrase.  \*/* final String KEY\_STORE\_PASSWORD = "changeit";   */\*\*  \* URL to connect to. That is, a server for which the above certificate is required.  \*/* final String URL = "https://localhost:8443/apn/swagger-ui.html#/";   public void sslConnect() throws KeyStoreException, IOException, CertificateException,  NoSuchAlgorithmException, KeyManagementException, UnrecoverableKeyException {  System.*setProperty*("javax.net.ssl.trustStore", "C:\\Nitesh-SENA\\ANTrackerWeb\\APN\\doc\\SSL\\server\\trustStore.jks");   *// Load the key store, containing the client-side certificate.* KeyStore keyStore = KeyStore.*getInstance*("pkcs12");  InputStream keyStoreInput = new FileInputStream(KEY\_STORE\_PATH);  keyStore.load(keyStoreInput, KEY\_STORE\_PASSWORD.toCharArray());  System.*out*.println("Key store has " + keyStore.size() + " keys");   *// Create an SSL context with our private key store.  // We are only loading the key-material here, but if your server uses a self-signed certificate,  // you will need to load the trust-material (a JKS key-store containing the server's public SSL  // certificate) as well.* SSLContext sslContext = SSLContexts.*custom*()  .loadKeyMaterial(keyStore, KEY\_STORE\_PASSWORD.toCharArray())  .useTLS()  .build();   *// SSLContext sslContext = SSLContext.getDefault();   // Prepare the HTTPClient.* HttpClientBuilder builder = HttpClientBuilder.*create*();  SSLConnectionSocketFactory sslConnectionFactory = new SSLConnectionSocketFactory(  sslContext, SSLConnectionSocketFactory.*ALLOW\_ALL\_HOSTNAME\_VERIFIER*);  builder.setSSLSocketFactory(sslConnectionFactory);  Registry<ConnectionSocketFactory> registry = RegistryBuilder.<ConnectionSocketFactory>*create*()  .register("https", sslConnectionFactory)  .register("http", new PlainConnectionSocketFactory())  .build();  HttpClientConnectionManager ccm = new BasicHttpClientConnectionManager(registry);  builder.setConnectionManager(ccm);   *// Perform a sample HTTP request.* try {  CloseableHttpClient httpClient = builder.build();  HttpGet httpget = new HttpGet(URL);  try {  CloseableHttpResponse response = httpClient.execute(httpget);  HttpEntity entity = response.getEntity();   System.*out*.println("----------------------------------------");  System.*out*.println(response.getStatusLine());  if (entity != null) {  System.*out*.println("Response content length: " + entity.getContentLength());  System.*out*.printf(EntityUtils.*toString*(entity));  }  EntityUtils.*consume*(entity);  } catch(Exception e) {  e.printStackTrace();  }  } catch(Exception e) {  e.printStackTrace();  }  }   public static void main(String[] args) throws Exception {  new ConnectIT().sslConnect();  } } |

(Code present in the Intellij)

Source

===============

https://medium.com/@niral22/2-way-ssl-with-spring-boot-microservices-2c97c974e83

https://wstutorial.com/rest/spring-boot-client-certificate.html

https://www.baeldung.com/x-509-authentication-in-spring-security

<https://dzone.com/articles/ssl-in-java>

https://blog.behrang.org/2019/01/30/ssl-mutual-authentication-java.html

Opened PostMan 🡪 Wrench Icon – Certificates

Added server.crt into the CA Certificates

And Client final certificate in the “Client Certificates”

