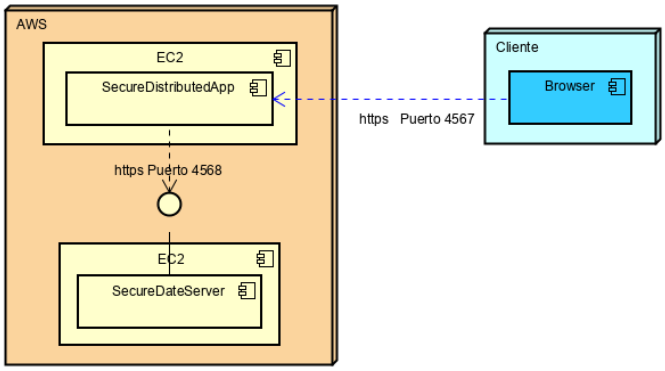
Aplicación distribuida segura en todos sus frentes

Amalia Inés Alfonso Campuzano - amalia.alfonso@mail.escuelaing.edu.co
Profesor Luis Daniel Benavides Navarro
Escuela Colombiana de Ingeniería Julio Garavito

I. Introducción

En este repositorio se desarrolló una aplicación en spark segura con autenticación, autorización e integridad de usuarios. Cuando se accede por el browser, se hace de forma segura. Adicionalmente, hace uso de otra aplicación con spark de la cual usa un servicio que devuelve la fecha.
Para garantizar autenticación se maneja registro e inicio de sesión de usuarios.
Para el desarrollo de esta arquitectura, se hace uso de un algoritmo hash, certificados almacenados en keystore (certificados propios) y trustore (certificados de sistemas en los que se confía) y dos máquinas virtuales EC2 de AWS, en donde se desplegarán las aplicaciones.

II. DIAGRAMA DE DESPLIEGUE



En la primera parte, el cliente se comunica con el servidor por medio de https por el puerto 4567. Para garantizarle al cliente autenticación se maneja registro e inicio de sesión de usuarios.

Al registrarse el cliente como un usuario, la aplicación guarda un código hash de la contraseña, haciendo uso del algoritmo SHA para asegurar integridad.

Cuando desea iniciar sesión, se verifica que el código hash de la contraseña ingresada, generado con el mismo algoritmo, sea igual al código guardado al registrarse. De esta manera se garantiza la autorización, para que solo pueda ver el perfil el usuario autorizado.
Además, el cliente puede solicitar la fecha ingresando la dirección, ya sea local o en AWS, seguida por "/date". La aplicación web al recibir esta solicitud del cliente, utiliza el otro servidor por medio de https por el puerto 4568 para hacer uso de sus servicios.

III. DESARROLLO

Parte 1: Local

En la primer parte, se implementaron y probaron las aplicaciones localmente.

• En la aplicación principal, SecureDistributedServer, Se establecieron las peticiones con spark, y se generó y configuró la keystore con el certificado para ingresar por https.

```
public static void main(String[] args) {
    port(getPort());
    staticFiles.location("/static");
    secure("deploy/keystoreapp.jks", "secureApp", "deploy/truststoreapp.jks", "secureAppT");
    //get("/login", (req, res) -> "Hello Secure World");

    get("/", (request, response) -> {
        response.redirect("index.html");
        return null;
    });

    get("/login", (request, response) -> {
        return login(request, response);
    });

    get("/date", (request, response) -> {
        return getDateOfServer();
    });

    post("/register", (request, response) -> {
        register(request);
        response.redirect("index.html");
        return null;
    });
}
```

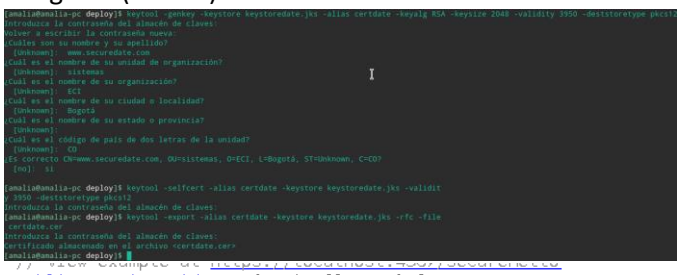
```
[amalia@amalia-pc deploy]$ keytool -genkey -keystore keystoreapp.jks -alias certapp -keyalg RSA
-keysize 2048 -validity 3950 -deststoretype pkcs12
Introduzca la contraseña del almacén de claves:
Volver a escribir la contraseña nueva:
¿Cuáles son su nombre y su apellido?
  [Unknown]:  www.distributedApp.com
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  Sistemas
¿Cuál es el nombre de su organización?
  [Unknown]:  ECI
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Bogotá
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  CO
¿Es correcto CN=www.distributedApp.com, OU=Sistemas, O=ECI, L=Bogotá, ST=Unknown, C=CO?
  [no]:  si

[amalia@amalia-pc deploy]$ keytool -selfcert -alias certapp -keystore keystoreapp -validity 3950
-deststoretype pkcs12
error de herramienta de claves: java.lang.Exception: El archivo de almacén de claves no existe:
keystoreapp
[amalia@amalia-pc deploy]$ keytool -selfcert -alias certapp -keystore keystoreapp.jks -validity
3950 -deststoretype pkcs12
Introduzca la contraseña del almacén de claves:
[amalia@amalia-pc deploy]$ keytool -export -alias certapp -keystore keystoreapp.jks -rfc -file certapp.cer
Introduzca la contraseña del almacén de claves:
Certificado almacenado en el archivo <certapp.cer>
[amalia@amalia-pc deploy]$
```

Imágen 1 - Configuración de certificados de SecureDistributedApp

• Para el registro del cliente, se utilizó un algoritmo que genera código hash llamado SHA, el cual puede encontrar en http://oliviertech.com/es/java/generate-SHA256--SHA512-hash-from-a-String/.

Cómo se explicó anteriormente, este código se genera al registrarse un usuario, y se guarda como su contraseña. Al iniciar sesión, se genera el código de la contraseña ingresada y se compara con el guardado.

• En el servidor externo, del que se utilizará un servicio SecureDateServer, también se generó un keystore.
Al exportar su certificado se pudo guardar en el truststore de la aplicación principal, para indicar que es un servidor en el que confía, cómo se puede ver en la Imágen 1 (línea 4).



```
public static void main(String[] args) {
    port(getPort());
    secure("deploy/keystoredate.jks", "secureDate", null, null);
    get("/", (request, response) -> {
        java.util.Date fecha = new Date();
        return fecha;
    });
}
```

• Este servicio se usa cuando se inicia sesión para mostrar la fecha en el perfil o cuando se ingresa por la dirección de la página principal: https://localhost:4567/date.

Para poder utilizar este servidor, se requiere establecer un método estático, ya que sin este se genera un error por el uso de los certificados. En este método se debe indicar que se conectará por localhost, y cuando se despliegue el servidor SecureDateServer, se debe cambiar por el nombre de la respectiva instancia.

```
static {
    //for localhost testing only
    javax.net.ssl.HttpsURLConnection.setDefaultHostnameVerifier(
            new javax.net.ssl.HostnameVerifier() {

        public boolean verify(String hostname,
                javax.net.ssl.SSLSession sslSession) {
            if (hostname.equals("localhost")) {
                return true;
            }
            return false;
        }
    });
}
```

En la siguiente imagen se muestra el código que permite el uso de SecureDateServer:

```
private static String getDateOfServer() throws IOException {
    String date = "";
    try {
        URL url = new URL("https://localhost:4568");
        BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()));
        String dataLine = "";
        while((dataLine = reader.readLine()) != null) {
            date += dataLine;
            //System.out.println(urlData);
        }
        System.out.println("Fin main :" + date);
    } catch (MalformedURLException ex) {
        Logger.getLogger(Spark.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Spark.class.getName()).log(Level.SEVERE, null, ex);
    }
    return date;
}
```

Al ejecutar la aplicación, se mostraba un error que indicaba que el certificado usado no estaba registrado en los certificados de java.

```
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException
```

Para solucionar este problema, se utilizó el siguiente comando:

keytool -importcert -trustcacerts -file certdate.cer -alias certdate -keystore "/usr/lib/jvm/java-8-openjdk/jre/lib/security/cacerts"

Al ejecutar este comando se requiere una contraseña que siempre es la misma: *changeit*

Para encontrar la ruta de la carpeta java (*/usr/lib/jvm/java-8-openjdk/jre/lib/security/cacerts*), se utilizó el comando *whereis java*.

Una vez hecho esto, ya se pudo acceder localmente.

Parte 2: Despliegue en AWS

•Una vez creadas las dos instancias, se actualizó la versión de java y se configuró los puertos por los que se puede acceder por https en las reglas de entrada del grupo de seguridad de cada instancia.

• Después se subió el .jar del servicio externo, SecureDateServer junto con la carpeta que contiene los certificados (carpeta deploy) en la raíz de la instancia, puesto que el jar no los almacena.

• Antes de generar el .jar de la aplicación principal, SecureDistributedServer, se modificaron dos partes del código de la aplicación principal.
Primero se modificó el método estático que permite la comunicación con el otro servidor, cómo se mencionó en la parte 1, cambiando localhost por el DNS de la instancia de SecureDateServer.

```
static {
    //for localhost testing only
    javax.net.ssl.HttpsURLConnection.setDefaultHostnameVerifier(
            new javax.net.ssl.HostnameVerifier() {

        public boolean verify(String hostname,
                javax.net.ssl.SSLSession sslSession) {
            if (hostname.equals("ec2-54-86-75-59.compute-1.amazonaws.com")) {
                return true;
            }
            return false;
        }
    });
}
```

Adicionalmente, se cambió la url en el método que hace uso del otro servidor:

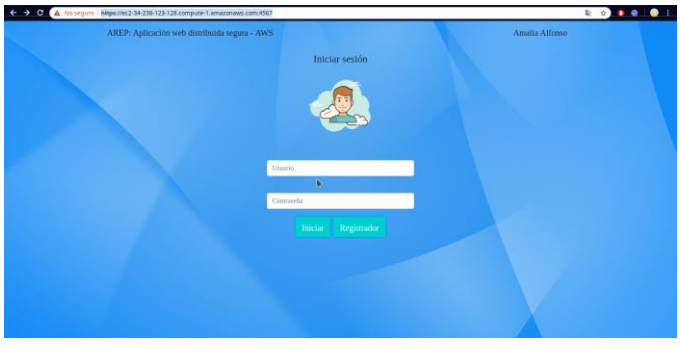
```
private static String getDateOfServer() throws IOException {
    String date = "";
    try {
        URL url = new URL("https://ec2-54-86-75-59.compute-1.amazonaws.com:4568/");
        BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()));
        String dataLine = "";
        while ((dataLine = reader.readLine()) != null) {
            date += dataLine;
            //System.out.println(urlData);
        }
        System.out.println("Fin main :" + date);
    } catch (MalformedURLException ex) {
        Logger.getLogger(Spark.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Spark.class.getName()).log(Level.SEVERE, null, ex);
    }
    return date;
}
```

• Una vez se genera y se sube el .jar con la carpeta de sus certificados, se realiza lo mismo que en la primera parte para validar el certificado en java.
En el comando se modificó la ruta del archivo que valida los certificados, puesto que en la instancia es diferente.

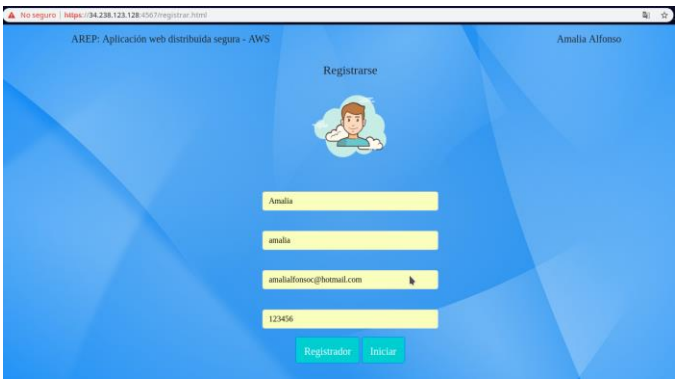
```
[amalia@amalia-pc AREP]$ ssh -i "keyapp.pem" ec2-user@ec2-34-238-123-128.compute-1.amazonaws.com
Last login: Wed Oct 23 01:29:49 2019 from 186.83.75.47

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
4 package(s) needed for security, out of 10 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-88-229 ~]$ ls
deploy
SecureDistributedApp-1.0-SNAPSHOT.jar
SecureDistributedApp-1.0-SNAPSHOT-jar-with-dependencies.jar
[ec2-user@ip-172-31-88-229 ~]$ cd deploy/
[ec2-user@ip-172-31-88-229 deploy]$ ls
2-codigo-local-uso de otro servidor.png     certdate.cer                    truststoreapp.jks
2-codigo-para AWS-uso de otro servidor.png  keystoreapp.jks
certapp.cer                                 keystore y cert (3pasos).png
[ec2-user@ip-172-31-88-229 deploy]$ keytool -importcert -trustcacerts -file certdate.cer -alias
certdate -keystore "/usr/lib/jvm/jre-1.8.0-openjdk/lib/security/cacerts"
Enter keystore password:
keytool error: java.lang.Exception: Certificate not imported, alias <certdate> already exists
[ec2-user@ip-172-31-88-229 deploy]$
```

• Finalizado el proceso, se probó su funcionalidad:
Ejecución de los archivos .jar

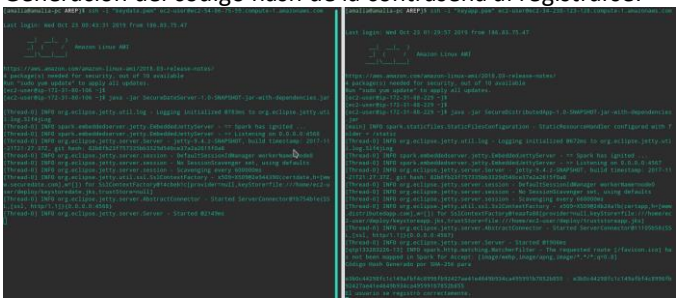
SecureDistributedServer



SecureDistributedServer-Registro



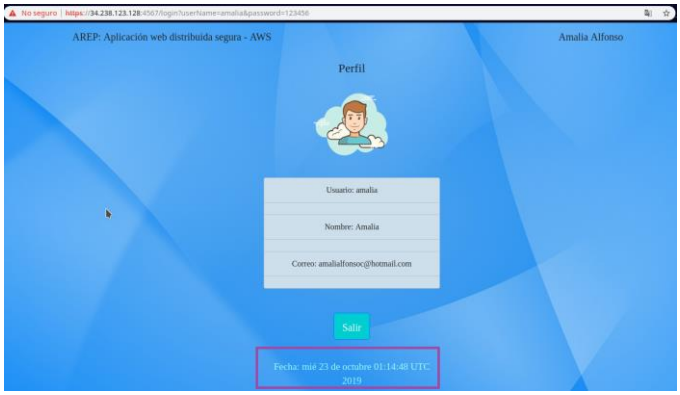
Generación del código hash de la contraseña al registrarse:



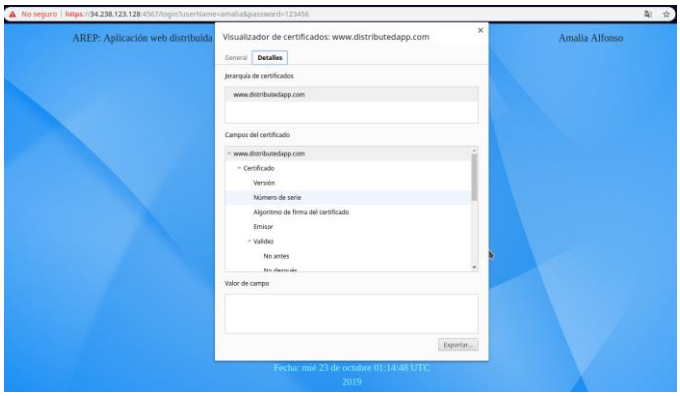
SecureDistributedServer - Inicio de Sesión

Comparación de códigos hash

SecureDistributedServer – Perfil



SecureDistributedServer - Ceritficado



Resumen Comandos:

Creación de certificado y keystore de SecureDistributedApp:

- `keytool -genkey -keystore keystoreapp.jks -alias certapp -keyalg RSA -keysize 2048 -validity 3950 -dname "CN=www.distributedapp.com, OU=Sistemas, O=ECI, L=Bogotá, ST=Unknown, C=CO" -deststoretype pkcs12`
- `keytool -selfcert -alias certapp -keystore keystoreapp.jks -validity 3950`
- `keytool -export -alias certapp -keystore keystoreapp.jks -rfc -file certapp.cer`

Creación de certificado y keystore de SecureDateServer:

- `keytool -genkey -keystore keystoredate.jks -alias certdate -keyalg RSA -keysize 2048 -validity 3950 "CN=www.securedate.com, OU=Sistemas, O=ECI, L=Bogotá, ST=Unknown, C=CO" -deststoretype pkcs12`
- `keytool -selfcert -alias certdate -keystore keystoredate.jks -validity 3950`
- `keytool -export -alias certdate -keystore keystoredate.jks -rfc -file certdate.cer`

Generar trustore en ServerDistributedApp:

- `keytool -importcert -alias certapp -file certdate.cer -keystore truststoreapp.jks`

Validar Certificado en java:

- `sudo keytool -importcert -trustcacerts -file certdate.cer -alias certdate -keystore "/usr/lib/jvm/java-8-openjdk/jre/lib/security/cacerts"`

  Contraseña: `changeit`

Para encontrar la carpeta en el que está java:

- `whereis java`

IV. ESCALABILIDAD

¿Cómo podemos escalar la arquitectura?

Para poder incorporar nuevos servicios se pueden utilizar otros servidores haciendo uso de microservicios. De manera en que SecureDistributedServer se pueda conectar a esos servidores y use de sus servicios, teniendo en el truststore los respectivos certificados para tener claridad de los servidores en los que confía y así conservar la seguridad y no modificar el código existente.

V. REFERENCIAS

- IT-Swarm.Net, & Akash5288. (2016, 22 diciembre). Trust Store vs Key Store - creando con keytool. Recuperado de https://www.it-swarm.net/es/java/trust-store-vs-key-store-creando-con-keytool/972567478/

- More of Less, & Steve Claridge. (2018, 12 septiembre). Diferencia entre un Java KeyStore y un TrustStore. Recuperado de https://www.moreofless.co.uk/java-keystore-truststore-difference-certificate/

- Código de registro. (2018, 23 diciembre). Error de Keytool de Java después de importar el certificado. Recuperado de https://codeday.me/es/qa/20181223/45749.html