Escuela Técnica Superior de Ingeniería Informática

Grado de Ingeniería Informática – Ingeniería del Software

PAI 3 - INSEGUS

Control de versiones

Versión	Realizado	Revisado	Descripción	Fecha
V1.0	Álvaro	Álvaro	Primera versión	05/04/2020
V2.0	Antonio	Carmen	Revisión de errores	06/04/2020

CONTENIDO

ÍDICE DE IMÁGENES

1 RESUMEN

En el presente documento se plantea dar soporte al problema planteado mediante la construcción de un mecanismo de comunicación bajo el uso del protocolo TLS (en su versión 1.3) así como el uso de “cipher suites” sobre este. En concreto, nuestro sistema es capaz de proveer a un cliente una solución por la cual este último es capaz de conectarse a otro equipo (dados datos previos como IP y socket), establecer un proceso de login (usuario y contraseña) y posteriormente realizar el envío de un mensaje decido por parte del usuario y almacenado localmente por el servidor (siempre y cuando las credenciales hayan sido correctas).

Todo este procedimiento se realiza por procedimientos seguros como serán explicados en los siguientes apartados.

La arquitectura detallada sería en el caso de hacerlo en dos máquinas distintas, para hacerlo en local valdría con usar el mismo keyStore en la misma máquina y no haría faltar exportar el certificado y crear un trustStore desde la parte del cliente.

2 ARQUITECTURA CLIENTE-SERVIDOR SEGURA

A continuación, se procederá a explicar el procedimiento seguido para la configuración tanto del servidor como del cliente. Destacar, que el proceso explicativo se llevará a cabo desde la perspectiva de un equipo con el sistema operativo WINDOWS 10 y con la versión de JAVA 13.0.2.

2.1 SERVIDOR

El primer paso que realizar es la construcción de una “*keyStore*” (almacén de claves), de necesaria existencia según la especificación SSL/TLS. En esta ocasión se ha optado por la utilización de la herramienta “*keytool*” proporcionada por Java. Como configuración añadida, se ha usado además el algoritmo RSA para su construcción, ya que por defecto es utilizado *“DSA”* el cual es incompatible con la versión de TLS utilizada en esta solución (1.3).

En concreto, el comando necesario para la construcción de la “keySotre” es (mediante la términal de Windows):

```
Keytool -genkey -keystore c:\SSLStore -alias SSLCertificate -keyalg RSA
```

Como parámetros observamos “*c:\SSLStore*” donde queda indicado el nombre y el lugar donde será almacenado el “*keyStore*”, “*SSLCertificate*” que indica el alias que tomará la “*keyStore*” creada y “*RSA*” que indica el algoritmo utilizado para su creación que sustituirá a *“DSA”* como indicamos anteriormente.

Durante la ejecución del comando se nos pedirán diversos datos, de los cuales tendremos que recordad con especial interés la contraseña que se nos solicita.

El próximo paso es la creación de un certificado el cual es requerido para el "*trueStore"* del cliente. Para poder obtenerlo también se va a utilizar la funcionalidad "*keytool" de Java. El comando utilizado para su generación es:*

```
keytool -export -alias SSLCertificate -keystore SSLCertificate public.cert
```

Donde hay que indicar el alías utilizado, así como el nombre que nombre que tendrá una vez exportado, *"public.cert"* en nuestro caso. Una vez generado se procederá a cedérselo al cliente y él será el encargado del todo el procedimiento de configuración del *"trueStore"* el cual queda explicado en el siguiente apartado.

Una vez generados todos los archivos anteriores se procede a compilar los archivos y ejecutarlos.

El primer paso será configurar el puerto en el que se realizarán las comunicaciones (tenga en cuenta a la hora de realizar las configuraciones que el cliente debe utilizar el mismo puerto que el servidor). Una vez terminadas las configuraciones el próximo paso es compilar el archivo, para ellos usaremos la terminal de Windows y nos tendremos que dirigir a la carpeta donde encuentre el archivo. A modo de ejemplo véase el siguiente comando:

```
cd Users\Alvaro\Desktop\ssii\PAI\pai3\src
```

Donde el archivo a compilar se encuentra dentro de una subcarpeta del escritorio del usuario Alvaro (Escritorio>ssii>PAI>pai3>src).

Una vez nuestra terminal este apuntando a esta carpeta procedemos a compilar el archivo mediante el comando:

```
javac BYODServer.java
```

Si no se ha producido ningún fallo, en la misma carpeta se generará el archivo *"BYODServer.class"* el cual será utilizado para ejecutar el sistema. Para ello podremos lanzarlo utilizando:

```
java –Djavax.net.ssl.keyStore=C:\SSLStore -
Djavax.net.ssl.keyStorePassword=PASSWORD BYODServer
```

En donde *"C:\SSLStore"* es la dirección donde se encuentra el *"keyStore"* y el parámetro *"PASSWORD"* deberá reemplazarse por el que ha sido utilizado cuando se generó en el primer paso de este apartado.

Como respuesta a la ejecución si todo ha ido bien se mostrará el mensaje *"Esperando conexiones de clientes..."* el cual indica que se queda a la espera de la conexión de un cliente. Sabremos que un cliente un cliente ha enviado un mensaje porque en una nueva línea nos aparecerá de nuevo el mensaje anterior. En esta ocasión, si nos dirigimos al archivo *"msg.txt"* podremos ver el mensaje que ha sido enviado.

2.2 CLIENTE

El primer paso desde la parte del cliente sería el de crear una *"keyStore"* y una "*trustStore"*, como se comentó en la parte del servidor se realizará la generación de

claves con RSA aunque por defecto se establezca DSA ya que no soportaría TLS 1.3 en el keyStore, procederíamos a través del siguiente comando:

```
Keytool -genkey -keystore c:\SSLStore -alias SSLCertificate -keyalg RSA
```

Tras rellenar los datos que nos piden y obtener dichos componentes procedemos a insertar el certificado que hemos obtenido del servidor con el siguiente comando:

```
Keytool -import -file C:\public.cert -alias firstCA -keystore myTrustStore
```

Donde *"public.cert"* sería el certificado del servidor, el alias sería para referenciar dicho certificado en el *"trusStore"* y con -keystore myTrustStore crearíamos el nuevo trustStore con dicho certificado y con el nombre anterior. Aquí se vería un ejemplo de la salida que conlleva en el terminal ejecutar dicho comando y los datos del certificado.

```
C:\>keytool -import -file C:\public.cert -alias firstCA -keystore myTrustStore
Enter keystore password:
Re-enter new password:
Owner: CN=Alvaro, OU=es, O=es, L=es, ST=es, C=es
Issuer: CN=Alvaro, OU=es, O=es, L=es, ST=es, C=es
Serial number: 88bae3a62be0a19d
Valid from: Sat Apr 04 20:01:27 CEST 2020 until: Fri Jul 03 20:01:27 CEST 2020
Certificate fingerprints:
         SHA1: 12:75:7E:E3:EC:75:D5:CB:B8:92:08:79:90:D7:32:88:5F:7C:E1:4F
         SHA256: 8E:D7:B4:A1:7A:64:8D:5D:76:09:42:53:45:C6:BA:0F:DD:98:63:52:34:D5:A8:B9:1E:65:D3:AA:BE:AC:8A:5A
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 28 94 C9 2C 65 EB 32 8A   26 3C 65 D3 2E 7C F9 EA  (..,e.2.&<e.....
0010: CA 72 D3 01                                        .r..
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\>
```

Ilustración 1: Insertar certificado

```
java –Djavax.net.ssl.trustStore=C:\myTrustStore -Djavax.net.ssl.
trustStorePassword=PASSWORD BYODClient
```

Con este comando se realizará el envío del mensaje al servidor, para ellos se nos pedirán tres datos, el usuario, la contraseña y el mensaje que se quiere enviar. Si el usuario y contraseña son correctos se procederá a almacenar dicho mensaje.

3 ANÁLISIS DE TRÁFICO DE RED EN COMUNICACIONES

Para realizar el análisis de tráfico de red se ha hecho uso de RawCap para interceptar todos los paquetes de información y almacenarlos en un archivo. Dicho archivo será abierto con la herramienta Wireshark, de esta forma podremos ver el intercambio de paquetes que se han producido.

A continuación, se mostrará un ejemplo de un intercambio de paquetes entre el servidor y el cliente, el cual hemos aplicado el filtro tcp.port == 7070, pues ese es el puerto que ambos comparten.

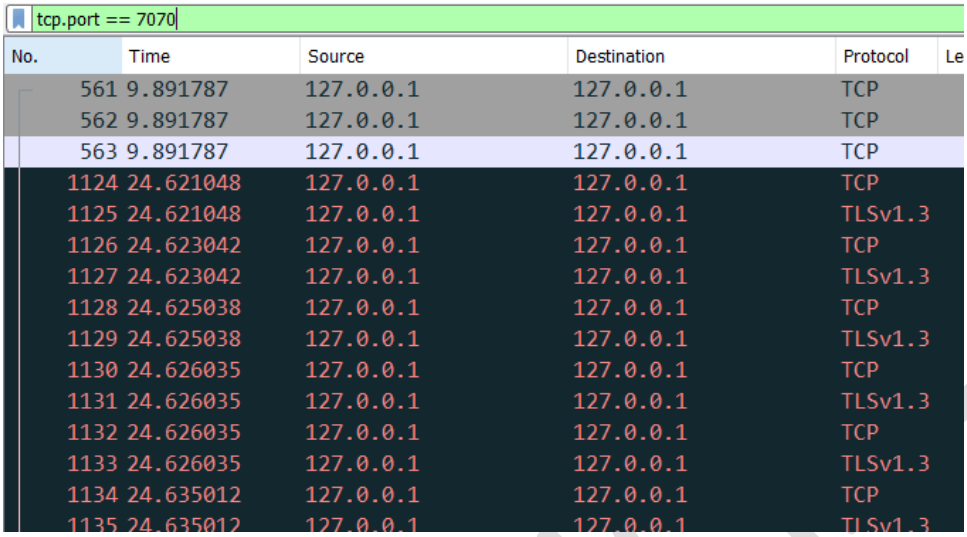

Ilustración 2: Captura Wireshark

Si accedemos clic derecho en una de las tramas y accedemos “Follow TCP Stream” podremos ver la comunicación realizada entre el cliente y el servidor, es decir, se podrá ver el nombre de usuario, la contraseña y el mensaje secreto, pero estarán codificados.

Si realizamos la prueba anterior sin tener en cuenta el SSL/TLS ni el cipher suites se puede ver que si accedemos a “Follow TCP Stream” podemos ver la comunicación establecida entre el servidor y el cliente sin ser cifrado.

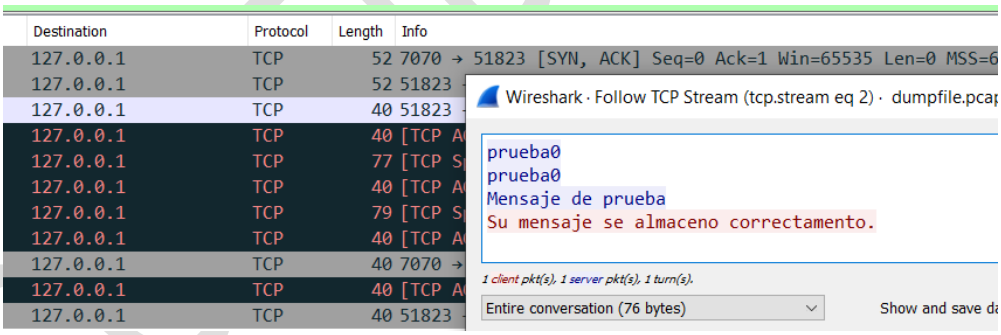

Ilustración 3: Follow TCP Stream

4 PRUEBAS DE RENDIMIENTO

Como requerimiento para el desarrollo de funcionalidad se solicita que el sistema desarrollado sea capaz de resistir el uso de aproximadamente 300 usuarios. Para ello se ha creado una clase auxiliar con el nombre BYODClient2.java el cual simula el envío de 300 mensajes por 300 usuarios diferentes.

De igual forma se usa un contador desde el primer envío de un mensaje por parte del usuario prueba0 | prueba0 hasta el último por parte del usuario prueba299 | prueba299.

El peor registro que se ha capturado ha sido de 51,43 segundos mientras que por otro lado el mejor caso ha sido 43,12 segundo. Para estas pruebas se han usados dos equipos en diferentes redes:

1. **Equipo 1 (servidor):** HP ay150ns con SSD Samsung Evo 850 y conexión simétrica de 100mb.
2. **Equipo 2 (cliente):** MSI Prestige 15 y conexión de 100mb de bajada y 50mb de subida.

Se han intercambiado los perfiles (cliente por servidor y servidor por cliente) y los resultados se han mantenido similares.

Como documento añadido se adjuntas dos capturas de tráfico, uno usando TLS en su versión 1.3 y los respectivos chipher suites (*"dumpfile.pcap"*) y otra sin el uso de estos *"dumpfile2.pcap" donde se pueden ver los datos de usuario y el mensaje*.