



Proyecto Seguridad en el diseño del Software

Nombre de la Aplicación: PedroAlbertoRFS (Remote FileSystem)

Luis Alfonso Jiménez Rodríguez - 48771949N

Juan García Martínez - 20085694R

Introducción

Nuestro objetivo con este proyecto es crear un sistema de almacenamiento, recuperación, consultado y eliminación, de manera remota de archivos. Tiene una seguridad lo suficientemente fuerte para que fuese difícil de atacar.

Para el intercambio de mensajería, el sistema está basado en sockets mediante el paso de estructuras, en el lenguaje de Go.

Hemos utilizado sockets por varias razones. La primera es porque es la arquitectura de transferencia de datos con la que más familiarizados estamos. Y la segunda es porque esta arquitectura permite bien el modelo de cliente-servidor que se nos pedía en este apartado.

El proyecto también tiene un sistema de creación de usuarios en el que permitimos al cliente crearse una cuenta de usuario e iniciar sesión con esta, teniendo un espacio seguro para guardar sus archivos.

Descripción

Funcionalidad

En una primera instancia, el usuario tendrá que registrarse o iniciar sesión para otorgar una privacidad a sus archivos subidos con respecto a los de el resto de usuarios. Durante el registro, se le pedirá al usuario un nombre que se usará para su inicio de sesión y una contraseña segura.

Una vez registrado, el usuario deberá iniciar sesión para acceder a la parte de la gestión de archivos.

Aquí, el usuario podrá realizar varias acciones:

- Consultar sus archivos
- Subir sus propios archivos (con un sistema de versionado)
- Descargarse sus archivos
- Eliminar sus archivos de su propio espacio.
- Cerrar sesión

No hemos considerado el compartir archivos entre usuarios para darle una mayor privacidad al usuario. (Se puede considerar en futuras versiones de la aplicación).

Diseño seguro

Sistema de usuarios:

En este aspecto hemos considerado una contraseña “segura” a una contraseña con 10 caracteres que contenga al menos 1 letra y 1 número.

También, estas son almacenadas de manera que, para transferirlas se hashean, y al entrar en el servidor, utilizamos PBKDF2 para volver a hashearlas y añadirle una sal (que se almacena en la base de datos).

Respecto a los datos de los usuarios estarán cifrados mediante AES empleando una clave privada en base a una parte de su contraseña.

Para que el servidor pueda reconocer a este usuario sin tener conocimiento sobre el cliente hemos empleado un token.

Transferencia:

Para la transferencia de archivos hemos usado la arquitectura de cliente-servidor con sockets mediante TLS que nos proporciona un protocolo orientado a garantizar una seguridad mediante conexiones TCP/IP.

Vimos que era una opción segura ya que nos permite crear canales con autenticación, confidencialidad, integridad y compresión de los datos, esta última para una mayor velocidad en la transmisión de estos.

También cabe destacar que es el protocolo recomendado actualmente.

Gestión de archivos:

Para el sistema de cliente-servidor, queríamos que en un principio el servidor tuviese conocimiento cero de la información que se almacena desde el cliente. Es decir, usuarios, contraseñas, nombre de los archivos, contenido etc... Todo esto, está cifrado en la propia base de datos (a la cual solo el servidor tiene acceso).

El sistema de almacenado lo hemos usado a través de sqlite3, almacenando toda esta información en una base de datos relacional con distintas tablas, algunas de ellas relacionadas entre sí mediante claves ajenas con borrado en cascada.

Las tablas son:

- Usuarios
- Sal (para las contraseñas)
- Archivos (junto con su información)

Para que el servidor detectase qué archivos son de cada usuario, en la propia sesión se almacena un token relacionado con el id del usuario, para que al almacenar el archivo, este esté relacionado con el usuario que ha almacenado ese mismo archivo.

file		user		salt	
id 🔑	#	id 🔑	#	userId 🔑	#
userId 🔑	abc	name	abc	salt 🔑	?
name	0110	password	abc		
nameHashed	0110				
peso	#				
version	#				
content	0110				

Planificación

Nuestra planificación ha sido muy simple y completa. La primera semana que nos reunimos escogimos un día en concreto para que ambos, si o si, estuviésemos libres.

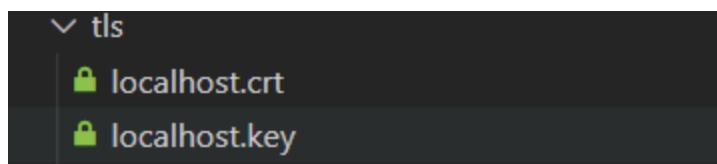
Este día acordado fueron los Viernes a las 17 mediante el uso de Discord. Creamos un canal privado en el cual íbamos compartiendo y recopilando información sobre go, sockets en go, y seguridad en go. Las llamadas solían durar mínimo 2 horas.


Respecto al código teníamos un repositorio remoto en GitHub llamado Remote-File-System donde ambos podíamos tocar código sin que le afectase al proyecto del otro.

Resultados

Implementación

Como hemos comentado anteriormente, nuestra arquitectura se basa en cliente-servidor con sockets empleando como modo de seguridad en la transmisión de los datos el protocolo tls. Para ello, necesitamos generar un certificado con su clave que la hemos almacenado en la carpeta llamada "tls".





Una vez que teníamos claro cómo realizar la transferencia, nos faltaba crear o utilizar un protocolo de comunicación. Inicialmente creamos uno propio que lo gestionamos mediante “splits” con símbolos reservados, esto nos llevó a un problema y era que el cifrado, podía generar esa combinación de símbolos cuando el fichero a cifrar era medianamente grande.

Para resolver esto, contactamos con el cliente y asesor profesional del proyecto ([Rafael](#)) y nos aconsejó emplear estructuras de datos que mediante una codificación no se pudiese generar esos problemas (función “encode” y “decode” de la librería JSON)

Para cifrar estos datos de los archivos empleábamos un cifrado AES con “CBCEncrypter” junto con “PKCS7Padding”, que actualmente no es correcto para el encriptado de archivos de tamaño dinámico. Como nos dimos cuenta antes de que pasase a producción gracias a nuestro asesor, lo cambiamos por el mismo cifrado AES pero que emplease “CTREncrypter” junto con un “iv” que estuviese almacenado en la el propio mensaje encriptado.

Para el nombre de estos ficheros también hemos empleado AES para devolverlo al cliente (pueda descargar/visualizar con el nombre real del archivo) y la función “hash” para poder hacer las consultas a la base de datos (poder comparar los nombres de los archivos sin necesidad de desenscriptar y así mantener el conocimiento cero en el servidor y disminuir el tráfico con el cliente).

Respecto al servidor, solo gestiona (almacena, borra, consulta) los datos recibidos por el cliente, y no los trata ya que no conoce los contenidos de estos. Solo se encarga de gestionar la conectividad a la base de datos como el versionado de los archivos, comparando los nombres de los archivos para almacenar.

Expectativas de seguridad y privacidad

Nuestro principal objetivo en la seguridad y privacidad era que el servidor fuese de conocimiento cero. Esto permite tres cosas:

- Que el usuario tenga su propia privacidad (claves privadas).
- Que sus archivos no se estén viendo comprometidos frente a un mal uso de su información privada.
- Que frente a un atacante malicioso que quiera robar los datos, estos datos tengan una fortaleza.

Incluso pudiendo robar la sesión con un token duplicado de un usuario, al no las claves con las que está cifrada la información, tampoco se podría descifrar correctamente esa información (sí que se podría consultar todos los datos pero no podía conocer la información que contiene ya que requiere tener la clave privada del usuario).

Otro aspecto importante respecto a la seguridad es emplear hash en información que no requiera ser rescatada. De esta forma, nunca se podría saber el contenido de estos datos. La hemos empleado para las contraseñas y para poder consultar los nombres de los archivos. Y esto último sí que es verdad que puede ser un tanto cargante para la base de datos ya que estaríamos duplicando un campo.

Al ser una empresa reducida económica y personalmente, somos más susceptibles a un ataque DDoS ya que no poseemos un servidor con una gran capacidad (pero sí suficiente). Para ello, tenemos pensado guardar “logs” para así detectar anomalías en las conexiones y poder reducir estos tipos de ataque.

Aún estamos contemplando el uso de firma digital para así otorgar una mayor veracidad a los archivos.

Conclusión

Gracias a todo lo anterior, hemos podido conocer las diversas formas de fortalecer la seguridad de una aplicación y así poder crearla.

Para ello nos hemos tenido que nutrir de información y nos hemos dado cuenta que internet no es una fuente fiable para la seguridad ya que hay mucho contenido desactualizado o poco seguro/formado. Por ello, nos decantamos por los apoyos obtenidos gracias a nuestro asesor profesional.

Como final, podemos reivindicar los aspectos que se podrían añadir a la aplicación en un futuro. Estos pueden ser la firma digital, una mejora al sistema de versionado permitiendo solo un número limitado de versiones, la posibilidad de compartir archivos entre usuarios (aunque así bajando un poco el nivel de privacidad) y mejorar el diseño de la aplicación.