Escuela Técnica Superior de Ingeniería Informática

Grado de Ingeniería Informática – Ingeniería del Software

PAI 2 - INSEGUS

Control de versiones

Versión	Realizado	Revisado	Descripción	Fecha
V1.0	Carmen	Álvaro	Añadido formato. Escrito apartados: resumen y árbol binario.	06/03/2020
V1.1	Álvaro	Antonio	Corrección de errores	07/03/20

ÍNDICE

ÍDICE DE IMÁGENES

1 RESUMEN

A lo largo de este documento se expondrá en procedimiento seguido para la realización del PAI 1, además de especificar la prueba realizado para verificar el funcionamiento del código.

Para simular los posibles casos de comprobación de integridad se propone al usuario la ejecución del archivo *"main.py"*. En este el usuario deberá indicar el caso que desea simular (existen tres), posteriormente indicar el nombre del archivo a buscar y finalmente proporcionar su token. A continuación, se deberá indicar si se desea utilizar los árboles previamente guardados o volver a generarlos. Una vez proporcionada toda la información, se buscará en los tres directorios el archivo que estamos buscando. Si el hash del archivo es igual al hash de los tres directorios sabremos que el archivo sigue integro y se le entregará al usuario la MAC. En el caso de que el hash del archivo no sea igual al hash de alguno de los tres directorios sabremos que el documento ha sido modificado en algunos directorios o en todos. En este caso, se generará un log indicando los directorios y el archivo afectado.

2 DESARROLLO

2.1 Árbol binario

Para la búsqueda de los ficheros en los distintos directorios, en este caso tres, se ha hecho uso de árboles binarios debido a su eficiencia de búsqueda.

Por cada directorio se crea un árbol. Este en el momento de su creación será serializado para poder se almacenado y se deserializará para realizar la búsqueda del archivo correspondiente según el hash que nos proporcione el usuario.

Para crear el árbol binario se accederá al directorio donde se encuentran los archivos y se seleccionará el documento del medio como la raíz del árbol. Una vez obtenida la raíz, está tendrá el primer archivo del directorio como hijo izquierdo y el siguiente archivo al de la raíz como hijo derecho.

Los hijos del hijo derecho de la raíz serán dos archivos que estén en el directorio entre el hijo derecho de la raíz y la raíz. Y de la misma forma, los hijos del hijo izquierdo de la raíz serán dos archivos que estén en el directorio entre el hijo izquierdo de la raíz y el último archivo del directorio.

La mecánica anterior se aplica para la construcción completa del árbol binario de un directorio. A continuación, se expondrá un ejemplo aclarativo:

File00.txt
File01.txt
File02.txt
File03.txt ← Raíz
File04.txt
File05.txt
File06.txt

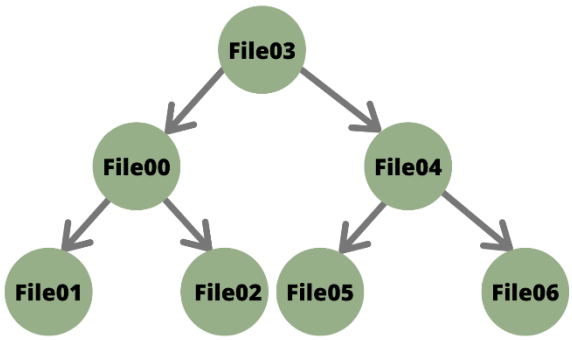

Imagen 1: Árbol Binario.

2.2 Verificación de integridad en servidor

Tal y como se muestra en el esquema propuesto, el proceso de verificación se puede diferenciar en dos fases (verificación de hashes y generación de MAC) de las cuales la primera repite en distintas rondas en función del número de servidores y/o las unidades de fallo de integridad detectadas y la segunda solo se realiza si pasa las pruebas de verificación de hashes en todos los servidores. El protocolo de verificación consta de los siguientes pasos.

2.2.1 Búsqueda del archivo

Dado un archivo X (p. e.: file1.txt) se procede a realizar su búsqueda en todos los árboles binarios. Una vez localizado en todos los archivos se crea la tupla <$File_{file1.txt}$, $Hash_{file1.txt}$> en donde $File_{file1.txt}$ hace referencia a la dirección física del archivo y $Hash_{file1.txt}$ al Hash que tiene almacenado el árbol binario del archivo en cuestión. Se crearán tantas tuplas como servidores existan.

2.2.2 Verificación HASH

Una vez obtenida la tupla <$File_{file1.txt}$, $Hash_{file1.txt}$> se procede a verificar la integridad del archivo volviendo a calcular hash de este. En concreto se utilizará SHA256 para su generación.

En el caso de que el proceso de comprobación fallase se generará un aviso en un archivo "*log*" recogiendo el fallo detectado (hora y localización del archivo). Si no existiese problemas en la verificación de los hashes se procedería a repetir el mismo proceso, pero en este caso repitiendo la ronda en otro servidor repitiéndose este protocolo hasta agotar todos los servidores.

Una vez comprobada la integridad de todos los ficheros, si en ningún servidor se ha producido error se procede a la generación de la MAC.

2.2.3 Generación de MAC

Tras la verificación de la integridad del archivo en el servidor se procese a la creación de la MAC del archivo sobre el que se está trabajando. Para su generación se ha optado por el uso de HMAC-SHA256 así como el uso del token compartido al inicio del proceso de verificación.

Al compartir tanto token como "*challenge*" en el proceso de generación de la MAC, el usuario podrá generarla la suya propia de manera local y verificar de igual forma la integridad de sus documentos. Se recuerda que la MAC solo se genera en el caso de que la integridad de los hashes no se haya visto afectada en ninguno de los casos de comprobación que han sido explicado en el apartado 2.2.2 – Verificación HASH.

2.2.4 Simulación consulta diaria

A pesar de que se nos indica en el enunciado la configuración de un protocolo generador de informes mensual se ha optado por repetir el proceso con una duración

en bucle de 10 segundos para demostrar su rápido funcionamiento. Simplemente basta con ejecutar el archivo *“main.py”* para desplegar todo el protocolo.

3 PRUEBAS

3.1 Test 1

HASH - test_integrity_true

Descripción: Dado un archivo y un hash (corresponde con el hash del archivo 1) volver a generar el hash del archivo y comprobar que es el mismo que el hash aportado.

Resultado: los hashes son iguales.

Conclusiones: el método creado funciona para el archivo comprobado en el caso positivo.

3.2 Test 2

HASH - test_integrity_false

Descripción: Dado un archivo y un hash (corresponde con el hash de un archivo diferente) generar el hash del archivo y comprobar que es distinto que el hash aportado.

Resultado: los hashes don diferentes.

Conclusiones: el método creado funciona para el archivo comprobado en el caso negativo.

3.3 Test 3

HASH - test_equals_hash_true

Descripción: dados los hashes de dos archivos iguales comprobar que sus hashes también son iguales.

Resultado: los hashes son iguales.

Conclusiones: el método creado funciona para el archivo comprobado en el caso positivo.

3.4 Test 4

HASH - test_equals_hash_false

Descripción: dados los hashes de dos archivos distinto comprobar que sus hashes también son distintos

Resultado: los hashes son distintos, se genera un archivo log.

Conclusiones: el método creado funciona para el archivo comprobado en el caso negativo.

3.5 Test 5

MAC - test_integrity_true

Descripción: dado un archivo, un token y una MAC (generada previamente sobre ese mismo archivo), generar una nueva MAC a partir del archivo y el token y compararla con la MAC dada.

Resultado: las MAC son iguales.

Conclusiones: el método creado funciona para el archivo comprobado en el caso positivo.

3.6 Test 6

MAC - test_integrity_diferent_file

Descripción: dado un archivo, un token y una MAC (generada previamente sobre otro archivo), generar una nueva MAC a partir del archivo y el token y compararla con la MAC dada.

Resultado: las MAC son distintas.

Conclusiones: el método creado funciona para el archivo comprobado en el caso negativo.

3.7 Test 7

MAC - test_integrity_different_user

Descripción: generar MAC a partir del mismo archivo, pero distintos tokens de usuario.

Resultado: las MAC son diferentes.

Conclusiones: el método creado funciona para el archivo comprobado en el caso negativo.

3.8 Test 8

MAC - test_integrity_method_bad_construct

Descripción: intentar generar una MAC y compararla sin dar otra MAC a comparar.

Resultado: nulo.

Conclusiones: el método es resistente a la mala construcción de este.

3.9 Test 9

INTEGRITY - test_integrity_true

Descripción: dado un archivo un token y un hash del servidor (hash generado previamente sobre ese mismo archivo) comprobar su integridad y generar una MAC en caso positivo, generar un log en caso negativo.

Resultado: se genera la MAC sobre el archivo.

Conclusiones: el método funciona en el caso positivo.

3.10 Test 10

INTEGRITY - test_integrity_false

Descripción: dado un archivo un token y un hash del servidor (hash generado previamente sobre otro archivo) comprobar su integridad y generar una MAC en caso positivo, generar un log en caso negativo.

Resultado: se genera un log de error sobre el archivo.

Conclusiones: el método funciona en el caso negativo.

3.11 Test 10

MAIN – Caso 1

Descripción: dado tres servidores cuya integridad no ha sido afectada (los tres servidores son clones de si mismo) se propone preguntar por la integridad de cualquier fichero. Ningún archivo se debería haber visto afectado.

Resultado: se devolverá la MAC del fichero.

Conclusiones: el método funciona en el caso positivo.

3.12 Test 11

MAIN – Caso 2

Descripción: dado tres servidores cuya integridad ha sido afectada (el archivo file81.txt ha sido modificado en el servidor 2) se propone preguntar por la integridad de fichero file81.txt.

Resultado: se genera un archivo en el log donde se indica el fallo de integridad para este archivo en el servidor 2. El proceso de identificación del fallo se repite cada 10 segundos.

Conclusiones: el método funciona en el caso negativo en el que solo se ve afectado un servidor.

3.13 Test 12

MAIN – Caso 3

Descripción: dado tres servidores cuya integridad ha sido afectada (el archivo file6.txt ha sido modificado en el servidor 1 y 2) se propone preguntar por la integridad de fichero file6.txt.

Resultado: se genera un archivo en el log donde se indica el fallo de integridad para este archivo en el servidor 1 y 2. El proceso de identificación del fallo se repite cada 10 segundos.

Conclusiones: el método funciona en el caso negativo en el que se ve afectado más de un servidor.

4 RESULTADO

Se ha simulado 3 servidores en la nube con 1000 ficheros cada uno, a su vez cada servidor tiene su árbol referenciado. El tiempo de búsqueda es prácticamente imposible de apreciar.

Como queda reflejado la última parte del apartado anterior (3 – Pruebas) se han generado los posibles errores de integridad que pueden aparecer en servidores de respaldo.

En el caso 1 no se muestra error sin embargo en los casos 2 y 3 se ha modificado un archivo aleatorio para comprobar que se es capaz tanto de localizar el mismo en los distintos servidores y árboles de respaldo como de señalar un fallo de integridad de manera individualizada por cada uno de los errores.

A modo de ejemplo se muestra una captura del funcionamiento estándar en el caso de detección de un error de integridad.

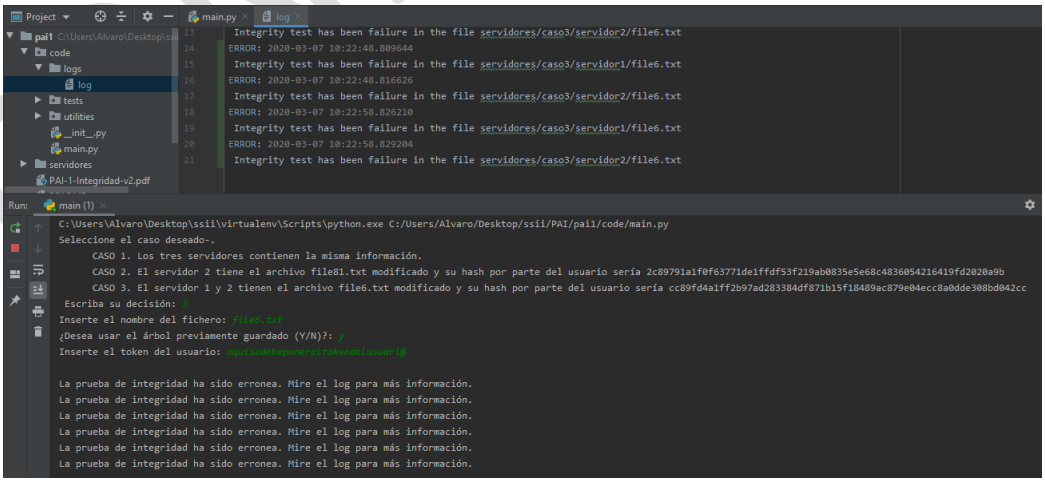

Imagen 2: Muestra de uso del protocolo.

Se ha preguntado por la integridad del archivo *"file6.txt"* el sistema, dado los árboles ya generados (tercera pregunta). El sistema ha detectado un error en los servidores 1 y 2.