HIDS: Almacenamiento masivo basado en verificadores de integridad

11-03-2022

Security Team 17

Responsables:
Jacobo García Velasco
José Carlos Morales Borreguero
Antonio Manuel Solís Miranda

Control de Versiones

Versión	Fecha	Descripción de la modificación
1.0	07/03/2022	Selección de estilos
1.1	08/03/2022	Comienzo en la redacción del documento
1.2	09/03/2022	Se introducen las pruebas realizadas
1.3	10/03/2022	Ampliado soluciones propuestas

Índice

Contenido

1. Resumen ejecutivo

En este informe se pretende aportar una solución al problema de control de la seguridad de la información presentado por la empresa consultante, que nos solicita la creación de un sistema de verificación de integridad de archivos. Para ello se trasladaron sus exigencias mediante la Política de Seguridad de la empresa, en la que pudimos destacar la petición de verificar los archivos de los sistemas críticos y dar un informe diario del resultado de este proceso.

Ante estas exigencias, la mejor opción en cuanto a precisión y escalabilidad fue la de un Host Intrusion Detection Systems, a partir de ahora HIDS. Esta solución basa su funcionamiento en un protocolo de prueba de posesión (Proof-of-Possession) definido por la dirección de INSEGUS y la que queda explicada de la siguiente manera:

En primer lugar, el cliente envía la dirección del archivo, el hash de este y un token, el servidor recoge esa información y busca en la base de datos la dirección del archivo, este proceso devuelve el hash del archivo original, por lo tanto, al compararlo en el siguiente paso tiene que dar que los hashes son los mismos, lo que significaría en un primer momento que el archivo no ha sido modificado, por último se calcula la MAC de ese hash, token y challenge (que en nuestro caso se ha usado una clave que solo la deberían de saber el cliente y el servidor) para enviarlo al cliente y que este compare si coincide con la MAC que él también ha conseguido de calcularla.

Para la comunicación entre cliente y servidor se ha utilizado tecnología socket, que nos permite mantener una conexión fluida y real entre dos entidades como son el cliente y servidor.

Como conclusión, la solución planteada cumple a la perfección con el problema a resolver de la manera más optima, segura y con un coste computacional mínimo, además cuenta con tecnologías de vanguardia, como puede ser Sha256 y HMAC, estándares de seguridad actuales y con gran reconocimiento.

A continuación, se ahondará en la tecnología usada mediante explicaciones técnicas y diagramas, así como una explicación más extensa del flujo de trabajo de la aplicación y la forma de transmisión de la información.

2. Tecnologías usadas

En cuanto a las tecnologías usadas podemos expresar que en el editor de código que nos hemos apoyado para la implementación a la solución del problema dado es Visual Studio Code. Nos hemos decantado por este entorno debido a su gran modularidad y personalización.

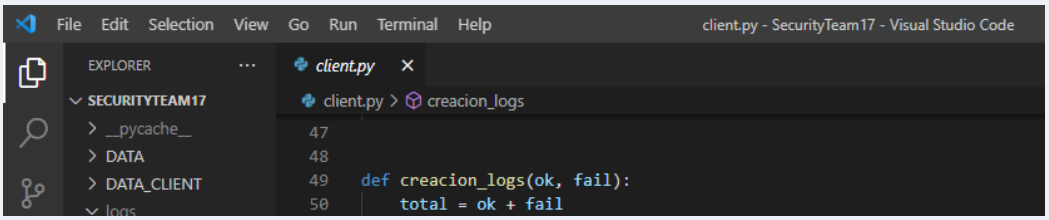

Ilustración 1: Visual Studio Code

El lenguaje de programación que hemos utilizado es Python en concreto todos los miembros hemos utilizado la versión 3.9.10. Nos hemos decantado por Python puesto que los integrantes del Security Team se siente mucho más cómodos en este idioma y además proporciona un amplio catálogo de librerías para el desarrollo de soluciones software en este ámbito.

***Ilustración 2**: Python*

Para alojar el código hemos usado GitHub ya que es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Este nos permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantener un seguimiento detallado de nuestro progreso. URL al repositorio: https://github.com/JoseCarlosMorales/SecurityTeam17

Hemos usado una estrategia de ramas simple en la que hemos creado una rama develop donde subíamos los cambios realizados durante el desarrollo, para mas tarde subir la versión final testeada y lista a master. Para ayudarnos en esta tarea, algunos componentes del equipo han utilizado git bash y otros han usado GitKraken, esta es una aplicación que usa git y que te brinda un entorno grafico que sirve de ayuda para el uso de los comandos, además de proporcionar una visión sencilla y cómoda de los commits realizados y como se están gestionando las ramas.

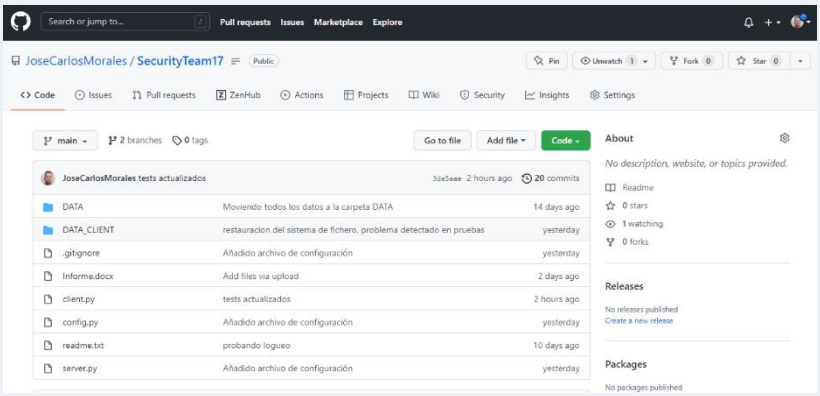

***Ilustración 4:** GitHub*

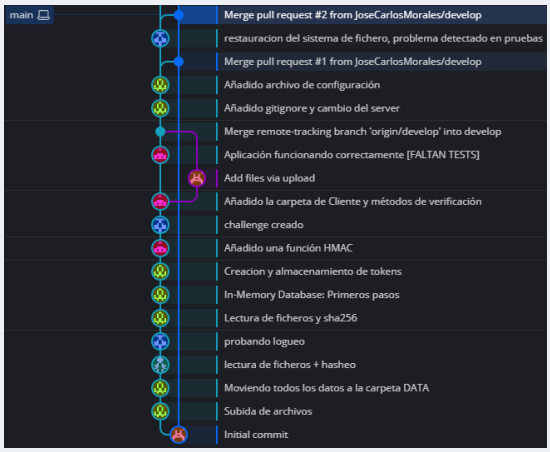

Ilustración 5: *GitKraken y la gestión de ramas*

Y, por último, para alojar y compartir este informe hemos utilizado OneDrive en el que nos aseguramos un control de versiones y que nos permite editar el documento simultáneamente varios miembros al mismo tiempo. A continuación, os mostraremos varias imágenes cómo prueba de que estas herramientas verdaderamente han sido utilizadas:

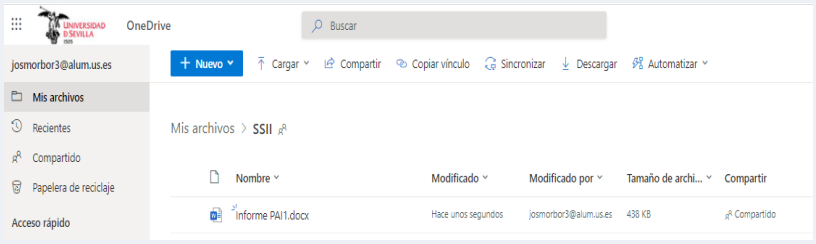

Ilustración 6: *OneDrive*

3. Soluciones aportadas

En este apartado se explicará con más detalle los entresijos de la solución, donde abordaremos los puntos sobre:

- Función hash
- Base de datos en memoria
- Uso de HMAC
- Comunicación mediante sockets

Para asegurar la integridad en el almacenamiento se ha usado una función secure hash, esta función hash funciona en una sola dirección, esto quiere decir que de cualquier contenido podemos generar su hash (su "huella dactilar digital") pero de un hash no hay forma de generar el contenido asociado a él, salvo probando al azar hasta dar con el contenido.

Entre las diferentes formas de crear hashes, el algoritmo que hemos usado en la solución propuesta ha sido SHA-256, uno de los más usados actualmente por su equilibrio entre seguridad y coste computacional de generación, pues es un algoritmo muy eficiente para la alta resistencia a ser descifrado.

Otra de las particularidades del algoritmo de hash SHA-256 es que la longitud del hash resultante es siempre igual, no importa lo extenso que sea el contenido que uses para generar el hash. El resultado siempre es una cadena de 64 de letras y números (con una codificación de 256 bits, 32 bytes), característica que nos ha sido de mucha ayuda para comprobar rápidamente si se estaba realizando correctamente el cifrado. Por todas estas particularidades comentadas, es por la que nos decidimos a usar este algoritmo hash para la codificación de nuestros ficheros.

Una de las primeras decisiones que tuvimos que enfrentar fue la de como almacenar los ficheros que servirán como verificadores de integridad para los archivos del cliente, después de ver distintas opciones como base de datos convencionales, en la nube o en memoria, nos decantamos finalmente por esta última, ya que cumplía a la perfección con las necesidades que se solicitaban.

La característica principal de este tipo de base de datos es almacenar toda la información en la memoria RAM y no en el disco, lo que nos brinda una velocidad de acceso superior a la da una base de datos tradicional puesto que el acceso a la memoria RAM se produce miles de veces más rápido que en el acceso a disco duro convencional. Esto también nos puede generar problemas pues no es muy escalable ya que dependemos de la cantidad de memoria RAM disponible, que suele ser de unos cuantos de gigas frente a Terabytes de las memorias de estado sólido y discos duros.

Para aplicar este tipo de base de datos hemos usado SQLite3 la cual permite el uso de bases de datos en memoria. Podemos destacar de la facilidad con la que se ha implementado la base de datos en memoria, simplemente llamando a la base de datos llamada ":memory:''. Al almacenarse todos los datos en la memoria RAM, cuando la base de datos se deje de utilizar, cuando se cierra la conexión entre usuario y servidor, y evidente cuando se apaga el dispositivo que aloja la base de datos, se elimina el contenido de la base de datos.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    bd = sqlite3.connect(':memory:')
    populate_database(bd, config.DIR_SERVER)
```

Ilustración 7: *Funcionamiento de HMAC*

Para asegurar la integridad en la transmisión del mensaje, principalmente en la comunicación entre el servidor y el cliente, se ha usado HMAC, mas concretamente HMAC-sha256. HMAC es una derivación de MAC (código de autenticación de mensaje) y que sirve para asegurar que algún sujeto intercepte el mensaje y lo cambie durante la transmisión, esto es gracias al uso de una clave secreta que solo saben el cliente y el servidor. Además, como ya hemos dicho, al tratarse de HMAC el mensaje queda encriptado mediante sha256, del que ya hemos hablado antes, y que nos proporciona una seguridad extra. Al usarse una clave secreta, se ha llegado a la conclusión que esta se transmitirá de forma física al cliente, para ahorra costes y prevenir posibles ataques que pongan en riesgos la integridad de la clave y por tanto del sistema.

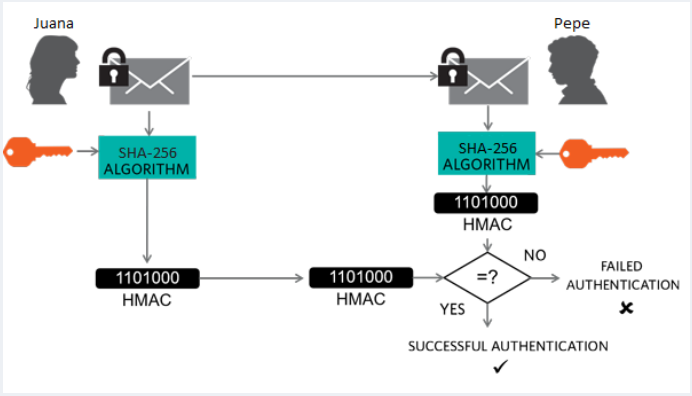

Ilustración 8: *Funcionamiento de HMAC*

Por último, destacar que se ha usado una conexión TCP por socket para la conexión entre el cliente y el servidor, esto no es mas que el servidor que se mantiene en espera hasta que el cliente se conecta y le envía la petición deseada, en nuestro caso y como se ha explicado en el resumen ejecutivo se le envía una la dirección del archivo, el hash de este y un token.

4. Diagrama y casos de uso

Caso de uso 1: Modificación de un documento

Si el usuario quiere comprobar si alguno de los documentos ha sido comprometido, este debe de enviar al servidor: la dirección del archivo, el hash de este y el token. Una vez que el servidor lo recibe busca en la tabla ”Ficheros” de la base de datos esa dirección y si lo encuentra (siempre lo debe de encontrar) nos devuelve el hash de ese mismo fichero que tenía la base de datos y que se entiende que es el documento que se tiene como referencia. Seguidamente se comparan los hashes y se comprueba sí coinciden, en caso de que no coincidan se devuelve al cliente el hash del archivo y un VERIFICATION_HASH_FAIL que por lo tanto al compáralo con la MAC del Cliente acabará en un INTEGRITY_FILE_FAIL.

Caso de uso 2: Sin modificaciones

En caso contrario (el archivo no ha sido modificado), el servidor con el hash, el token y el challenge, siendo este último una clave acordada entre el cliente y el servidor previamente y que nadie más debe de saber, calcula la MAC, en nuestro caso la HMAC con el algoritmo sha256. A continuación, la envía al cliente, con los mismos parámetros citados anteriormente este calcula la HMAC y la compara con la recibida. En caso de ser la misma, devuelve un INTEGRITY_FILE_OK, es decir, sin modificaciones.

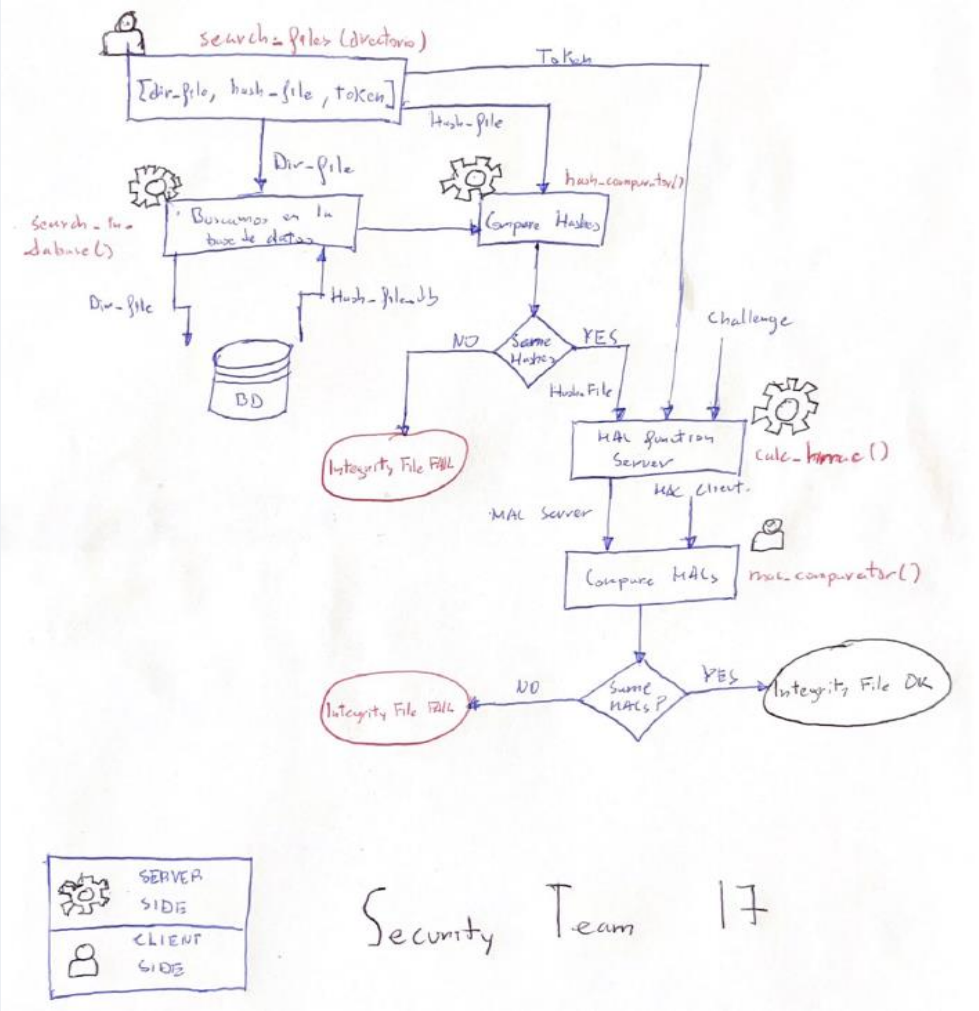

Ilustración 9: *Flujo de trabajo*

5. Pruebas realizadas

En este apartado vamos a comentar como hemos realizado las pruebas para garantizar que nuestro sistema funciona correctamente.

Para las pruebas se ha realizado la verificación de la integridad para los archivos diariamente durante un mes. Nuestro sistema aportará el porcentaje de archivos cuya integridad ha sido vulnerada.

Haremos uso de dos contadores, en el caso de que el archivo haya sido modificado sumaremos una unidad en el contador de los archivos catalogados como “fail” y en el caso de que los archivos no hayan sido modificados sumaremos una unidad en el contador de los archivos catalogados como “ok”.

Una vez que tenemos la información requerida que nos interesa procedemos a la construcción de unos logs que se producen automáticamente cuando el cliente se conecta al servidor y en el que recogemos la fecha de la creación del log, el porcentaje total de ficheros que no han sido modificados, el porcentaje total de ficheros que han sido modificados, el número total de ficheros y el número de ficheros que contabiliza el sistema que han sido modificados.

A continuación, mostraremos algunas capturas de las pruebas realizadas en las que hemos ido incrementando el número de fichero modificados para comprobar que la reacción del sistema a estos cambios eran los esperados.

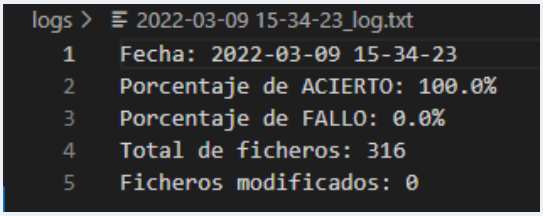

Ilustración 10: *Log generado sin modificar ningún fichero*

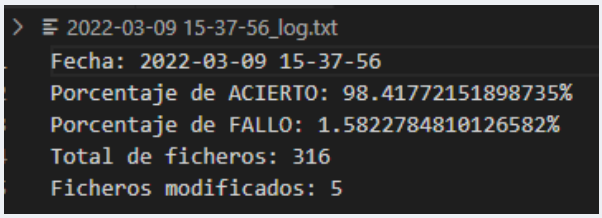

Ilustración 11: *Log generado modificando 5 ficheros*

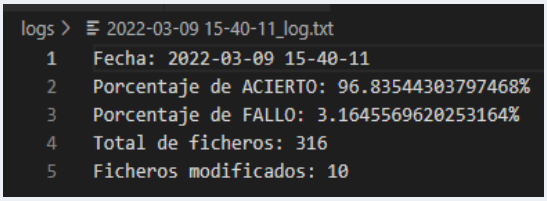

Ilustración 12: *Log generado modificando 10 ficheros*

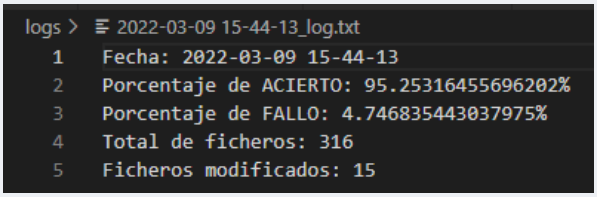

Ilustración 13: *Log generado modificando 15 ficheros*

Fuentes

[1] Wikipedia HMAC: https://es.wikipedia.org/wiki/HMAC
[2] Bit2Me Sha256: https://academy.bit2me.com/sha256-algoritmo-bitcoin
[3] Enseñanza Virtual Seguridad en Sistemas Informáticos y en Internet: https://ev.us.es
[4] Base de datos en memoria: https://bsginstitute.com/bs-campus/blog/Base-de-Datos-en-Memoria-1113