# PAI 1 - HIDS


**Fecha:** 13/02/2023

**Grupo de teoría:** 1
**Grupo de práctica:** Security Team 6
**Repositorio:** https://github.com/abernalmar/PAI-1-HIDS

**Miembros:**

Aguilar Román, Patricia | Correo: patagurom1@alum.us.es
Bernal Martín, Ángela | Correo: angbermar1@alum.us.es
Martín Sánchez, Paola | Correo: paomarsan@alum.us.es

# Índice

**Índice** **1**

**Introducción** **2**

**Objetivos del proyecto** **2**

**Decisiones importantes** **2**

**Soluciones adoptadas** **3**

**Implementaciones desarrolladas** **3**

Archivo save_files.py 3

Archivo main.py 4

Archivo report.py: Generación de reports diarios 5

Archivo report_mensual.py: Generación de reports mensuales 6

Pruebas de posesión (nubes no confiables) 6

**Resultado y análisis de pruebas** **7**

Pruebas de reports diarios 7

Pruebas de reports mensuales 7

Pruebas de posesión (nubes no confiables) 8

**Conclusiones** **9**

# Introducción

En este Proyecto de Aseguramiento de la Información (PAI) se pretende comenzar a familiarizarse con del trabajo en el gobierno/gestión/tecnologías de la seguridad de la información y en este caso de la verificación de la integridad de datos/información en un sistema informático de almacenamiento masivo simulado en un host local.

Cada día nos enfrentamos a un creciente número de intrusiones no autorizadas de personas ajenas a nuestras empresas que acceden a informes, archivos y datos, lo que se ha convertido en uno de los mayores desafíos para todas las empresas. Aunque muchas empresas han asignado un presupuesto anual para adquirir dispositivos y software que puedan proteger la información de la empresa, a medida que la seguridad se fortalece, los atacantes actualizan sus técnicas y encuentran nuevas formas de exponer la información de la empresa.

En respuesta a estas situaciones, es necesario implementar herramientas adicionales que puedan detectar cualquier acceso no autorizado a nuestro sistema o posibles anomalías en nuestros archivos informáticos y nos informen lo antes posible.

# Objetivos del proyecto

1. Desarrollar/Seleccionar el más conveniente HIDS basado en verificadores de integridad de acuerdo con lo exigido en la Política de Seguridad bien para nubes confiables o no confiables.

2. Desplegar el HIDS en un sistema de información simulado con miles de ficheros.

3. El proceso de verificación se realizará a intervalos, en este caso diariamente y debe almacenarse un informe de un mes entero, indicando el porcentaje de ficheros/directorios no íntegros con respecto al número total de ficheros/directorios que se verifica para cada día del mes y los nombres de los ficheros detectados como no íntegros en cada día del mes.

4. Se deberá evitar en la medida de lo posible las debilidades típicas de los HIDS.

5. Se debe razonar y demostrar la eficiencia de la solución aportada a la hora de localizar, calcular y almacenar la información en el HIDS.

# Decisiones importantes

A continuación, las decisiones que hemos considerado más importantes en cuanto al proyecto desarrollado:

- Hemos decidido utilizar Python, ya que es un lenguaje de programación sencillo que además ya hemos mantenido contacto con él anteriormente y por lo tanto nos resulta conocido.

- Otra decisión importante que tomamos fue usar GitHub para gestionar el código, creando un repositorio donde hemos podido compartir entre nosotras el código que se iba implementando del proyecto.
- Hemos decidido implementar pruebas de posesión en nubes no confiables, ya que durante la clase de seguimiento, el profesor nos explicó el funcionamiento de ambos métodos y nos pareció mejor usar el de nubes no confiables, ya que se nos explicó la metodología a seguir para llevarlo a cabo y nos resultó viable con respecto a nuestro proyecto.
- Hemos decidido que en 10 segundos vamos a simular 1 día entero y por lo tanto en 300 segundos simulamos un mes completo.
- Decidimos también que no nos era necesario volcar el diccionario que se almacena en memoria con todos los hashes en un archivo local para no tener que protegerlo, la idea de volcar los hashes que se almacenan en el diccionario era que estos no se tengan que recalcular cada vez que se inicia la aplicación para cuando fueran 1000, pero ahora que tenemos esos mil ficheros nos hemos fijado en que tarda menos de un segundo en calcularlos igualmente. Si lo hubiéramos almacenado en un archivo dentro del propio repositorio, habría que haberlo protegido calculando el propio hash de ese archivo.

# Soluciones adoptadas

Se busca implementar una solución de seguridad que permita al cliente realizar comprobaciones de los archivos de su sistema para detectar cualquier posible actividad sospechosa.
Hemos desarrollado una aplicación con una carpeta resources, en la cuál tenemos un listado de 1000 archivos. Nuestra aplicación genera los hashes de estos archivos y los almacena en un diccionario cuyas claves son la extensión y el nombre del archivo. Periódicamente, el sistema vuelve a generar los hashes de cada archivo y los compara con el almacenado, si el hash ha cambiado, el archivo ha sido modificado, por lo que guarda el cambio en un fichero *changes.log*.

Además, el sistema genera informes diarios y mensuales.
En los diarios, se informa de los nombres de los archivos que han sido modificados, junto a su fecha y el último hash calculado.
En los informes mensuales, aparece la misma información que en los diarios pero además se indica el número de archivos que han sido modificados en el mes y el porcentaje total.

Hemos implementado también pruebas de posesión para nubes no confiables. Estas pruebas verifican la autenticidad de un archivo en la nube mediante la comparación de un hash con un “challenge”, proporcionado por el cliente con un hash previamente almacenado en un diccionario.

# Implementaciones desarrolladas

## Archivo save_files.py

Este programa realiza una verificación de integridad de los archivos ubicados en el directorio "resources" y los almacena en un diccionario global llamado "DICC_HASH". El usuario debe

especificar el algoritmo criptográfico a utilizar mediante una entrada de consola. El programa implementa las siguientes funciones:

- *get_name(file):* recibe como parámetro el nombre del archivo con su extensión y devuelve el nombre sin extensión.

- *get_extension(file):* recibe como parámetro el nombre del archivo con su extensión y devuelve la extensión del archivo.

- *save_files():* recorre todos los archivos del directorio "resources", calcula su hash y los almacena en el diccionario global "DICC_HASH", utilizando la extensión del archivo como clave del primer nivel y el nombre del archivo sin extensión como clave del segundo nivel.

- *check_digest(file, dicc):* recibe como parámetros el nombre del archivo a verificar y el diccionario global que contiene los hash previamente calculados. La función compara el hash del archivo especificado con el hash almacenado en el diccionario. Si el hash es diferente, la función devuelve una lista que contiene la fecha y hora de la última modificación del archivo, el nombre del archivo y la cadena "File changed". Si el hash es igual, la función no devuelve nada.

- *remove_log_content():* borra el contenido del archivo "changes.log".

- *write_log(check_data):* escribe en el archivo "changes.log" una entrada que contiene la fecha y hora de la verificación, el nombre del archivo verificado y el resultado de la verificación (en caso de que el hash del archivo haya cambiado).

- *digest(path,alg):* recibe como parámetros la ruta del archivo a calcular el hash y el algoritmo criptográfico a utilizar. La función calcula el hash del archivo y devuelve su valor en formato hexadecimal.

En resumen, el programa realiza una verificación de integridad de los archivos en un directorio utilizando hash criptográficos y almacena los valores en un diccionario global. Además, guarda en un archivo de registro las verificaciones realizadas y también si algún archivo cambia su contenido.

## Archivo main.py

Este programa tiene como objetivo verificar la integridad de los archivos en un directorio mediante la comparación de su hash SHA-256 con el valor hash almacenado en un diccionario creado por la función *save_files()* del archivo *save_files.py*.

Se establece una función *sig_handler()* que maneja la señal de interrupción de Ctrl + C y finaliza el programa. Luego, se establece una señal de interrupción con signal.signal() para que la función *sig_handler()* maneje la interrupción si se recibe la señal SIGINT.

A continuación, se establece un tiempo inicial utilizando time() para medir el tiempo total de ejecución del programa. Luego, se llama a la función *save_files()* para crear el diccionario que contiene la información de cada archivo en el directorio y se guarda en la variable llamada dicc.

Después, se calcula el tiempo transcurrido desde el inicio de la ejecución utilizando la diferencia de *time()* del tiempo inicial y el tiempo actual y se imprime en la consola.

Luego, se establece la ruta actual utilizando os.path.dirname(os.path.abspath(__file__)) y se lista los archivos en el directorio */resources* con *os.listdir(current_path+"/resources")* y se guarda en la variable *FILES*.

A continuación, se define una función *timer()* en la que cada 2 segundos, se recorren todos los archivos en el directorio y para cada archivo, se llama a la función *check_digest()* para verificar si el archivo ha sido modificado desde la última comprobación. Si se detecta un cambio, se escribe un registro en el archivo de registro utilizando la función *write_log*.

La función *remove_log_content()* se llama antes de ejecutar la función timer() para borrar el contenido del archivo de registro (*changes.log*) antes de comenzar las comprobaciones.
Finalmente, se llama a la función *timer()* para iniciar la comprobación de integridad de archivos y luego se llama a las funciones *populate_html()* y *populate_html_mensual()* para generar los informes HTML con detalles de los cambios detectados en los archivos, si los hay. Esta función utiliza la función *create_table_html()* para crear la tabla HTML.
La función *populate_html()* se llama de forma recurrente y tiene un tiempo de espera de 20 segundos entre cada llamada (lo cuál sería equivalente a 1 día). Sin embargo, la función *populate_html_mensual()* tiene un tiempo de espera de 300 segundos entre cada llamada (lo equivalente a 1 mes).

## Archivo report.py: Generación de reports diarios

El código implementado para la realización de los reports diarios se encuentra en el fichero report.py. Este programa realiza una tarea de monitoreo de cambios en archivos y crea informes HTML cuando se detectan cambios.
A continuación, describimos detalladamente el funcionamiento de cada función en este fichero:

- *create_table_html(headers, report, data):* Esta función crea una tabla HTML con encabezados, la fecha del informe y los datos proporcionados. Los encabezados son una lista de cadenas que se utilizan para nombrar las columnas de la tabla. El parámetro report es una cadena que se utiliza para indicar la fecha del informe. El parámetro data es una lista de tuplas, donde cada tupla contiene los datos para cada fila de la tabla. La función devuelve una cadena que representa el HTML de la tabla.

- *populate_html():* Esta función utiliza un temporizador de subprocesos para ejecutarse cada 20 segundos. Lee los cambios del archivo *changes.log* que ocurrieron en los últimos 20 segundos y, si hay cambios, crea un archivo HTML en la carpeta reports que muestra una tabla con los detalles de los cambios. Los detalles se almacenan en una lista changes, que

contiene tuplas que contienen la marca de tiempo, el nombre del archivo y el último hash calculado. Para crear el archivo HTML, la función llama a *create_table_html()* con los encabezados adecuados y la lista de cambios. El archivo HTML se guarda con un nombre que contiene la fecha y hora actual, en la carpeta *reports*.

## Archivo report_mensual.py: Generación de reports mensuales

El código implementado para la realización de los reports mensuales se encuentra en el fichero report.py.

- *create_table_html(headers, report, data):* se encarga de crear una tabla HTML que se utilizará en los informes generados por el script. Toma tres argumentos: "headers" (los encabezados), "report" (el nombre del informe) y "data" (los datos que se mostrarán en la tabla). La función crea una plantilla HTML predefinida y luego agrega los datos proporcionados para generar la tabla.

- *populate_html_mensual():* esta función busca en el archivo llamado *changes.log* los cambios que han ocurrido en el último mes (300 segundos en nuestro caso) y, si hay cambios, crea un archivo HTML en la carpeta *reports_mensual* que muestra una tabla con los detalles de los cambios. La función utiliza la anterior definida (*create_table_html*) para generar la tabla y agrega algunos datos adicionales, como el número total de archivos modificados y el porcentaje de archivos modificados en relación al número total de archivos. La función utiliza un temporizador de 30 segundos para ejecutarse periódicamente.

## Pruebas de posesión (nubes no confiables)

Este código es una implementación de una función que verifica la autenticidad de un archivo en la nube mediante la comparación de un hash proporcionado por el cliente con un hash previamente almacenado en un diccionario. La función toma cuatro argumentos: un reto aleatorio "challenge", la extensión del archivo, el nombre del archivo y el hash proporcionado por el cliente.

El valor de "challenge" es un número aleatorio que da el cliente, en este caso, hemos hecho que sea un valor aleatorio. La extensión de archivo y el nombre de archivo se utilizan como claves para buscar el hash correspondiente en el diccionario de hashes previamente almacenados en la función *save_files()*. El hash del archivo que se proporciona se compara con el hash almacenado en el diccionario para realizar la prueba de posesión.

La función *verify_file_cloud()* toma los primeros tres dígitos del hash proporcionado por el cliente y los primeros tres dígitos del hash almacenado en el diccionario para el archivo correspondiente y los convierte en un número entero, para posteriormente sumarles el  valor del “challenge”.

Si la suma de ambos números es igual, se considera que la prueba de posesión es correcta y la función devuelve True. De lo contrario, devuelve False.

```python
def verify_file_cloud(challenge, extension, filename, proof):

    first_three_digits = proof[:3]
    summed_digits = int(first_three_digits, 16) + challenge
    challenge_response = int(hash_dict[extension][filename][:3], 16) + challenge

    # Verificar la respuesta
    if challenge_response == summed_digits:
        return True
    else:
        return False
```

# Resultado y análisis de pruebas

Para comprobar el funcionamiento del código implementado,  hemos realizado varias pruebas:

## Pruebas de reports diarios

Para comprobar el funcionamiento del código implementado hemos realizado una prueba en la que, con la aplicación en marcha, modificamos uno de los ficheros contenidos en resources. Por ejemplo, hemos modificado el archivo “text2.txt”, en el que hemos añadido: “, ya no”:

```
resources > text2.txt
  1  yo también soy un documento seguro, ya no
```

Como podemos comprobar, en 10 segundos o menos (lo equivalente a 1 día), se genera un informe en html en la carpeta reports. Al abrir este informe nos encontramos con la siguiente tabla:

**REPORT DATE: 21-Feb-2023-22-31-18.**

| Timestamp | File Name | Last Hash Calculated |
|---|---|---|
| 21-Feb-2023 (22:31:15) | text2.txt | f349464ac84710e4ac79cd69b28140336e5047cda8deee920f48c5cf877fb9fd |

En esta tabla podemos ver la fecha en la que ha sido modificado el archivo, el nombre del archivo modificado y el último hash calculado.
Podemos comprobar así que el código implementado para la generación de reports diarios funciona correctamente.

## Pruebas de reports mensuales

Para comprobar el funcionamiento del código implementado hemos realizado una prueba en la que, durante 5 minutos (300 segundos), lo equivalente a un mes, hemos realizado diversos cambios en distintos archivos de la carpeta resources.
Por ejemplo, hemos modificado seis ficheros pdf de la carpeta *resources.*

Además de generar los reports diarios similares al descrito en el apartado anterior, al cabo de 5 minutos debe generarse un informe en html en la carpeta reports_mensual. Al abrir este informe nos encontramos con lo siguiente:

**REPORT MENSUAL: 22-Feb-2023-18-09-48.**

| Timestamp | File Name | Last Hash Calculated |
|---|---|---|
| 22-Feb-2023 (18:08:03) | CAI1-Consulta 2 - copia (7) - copia.pdf | 8b2325e977127b64397ef5f6f9d0efb6a2e292d336b518326e7073047945a580 |
| 22-Feb-2023 (18:05:24) | CAI1-Consulta 2 - copia (7) - copia - copia.pdf | 4c46b96360d8ce0eb4582673e63425749cc520b8730f02a7ef6a5460fcdb6bf5 |
| 22-Feb-2023 (18:05:18) | CAI1-Consulta 2 - copia (6).pdf | 4c46b96360d8ce0eb4582673e63425749cc520b8730f02a7ef6a5460fcdb6bf5 |
| 22-Feb-2023 (18:05:10) | CAI1-Consulta 2 - copia (6) - copia.pdf | 4c46b96360d8ce0eb4582673e63425749cc520b8730f02a7ef6a5460fcdb6bf5 |
| 22-Feb-2023 (18:05:04) | CAI1-Consulta 2 - copia (6) - copia - copia.pdf | 4c46b96360d8ce0eb4582673e63425749cc520b8730f02a7ef6a5460fcdb6bf5 |
| 22-Feb-2023 (18:04:58) | CAI1-Consulta 2 - copia (5).pdf | 4c46b96360d8ce0eb4582673e63425749cc520b8730f02a7ef6a5460fcdb6bf5 |

Modified files: 6 (0.6% of total)

En esta tabla podemos ver las fechas y horas exactas en las que se modifica el fichero, el nombre de este y el último código hash calculado. Además, en la última línea se indica el número de ficheros que han sido modificados y el porcentaje total de archivos modificados con respecto al total existentes.
Por lo tanto, podemos demostrar así que el código implementado para la generación de reports mensuales funciona correctamente.

## Pruebas de posesión (nubes no confiables)

Para realizar las pruebas de posesión de nubes no confiables, hemos proporcionado dos ejemplos, uno en el que el cliente proporciona un hash correcto y otro con un hash incorrecto. Si el código implementado funciona, debe imprimir por pantalla un mensaje indicando si la prueba de posesión ha sido exitosa o no. En este caso, la primera prueba debe salir exitosa y la segunda no.

```
# Ejemplo buen hash
challenge = random.randint(0, 100)  #Genera un reto aleatorio
filename = 'text'
extension = '.txt'
proof = '403982b3df390ddbb963b681dd9c4d183a3b396f24f1c5d653da4e7ecd2357f0'  #Hash correcto del text.txt

print('Prueba de posesión para text.txt con hash correcto:')
if verify_file_cloud(challenge, extension, filename, proof):
    print("Prueba de posesión correcta")
else:
    print("Prueba de posesión ha fallado")
```

```python
# Ejemplo mal hash
challenge = random.randint(0, 100)  #Genera un reto aleatorio
filename = 'text'
extension = '.txt'
proof = 'a1b2c3d4'  #Este hash no es el correcto, por lo que la respuesta será: La prueba de posesión ha fallado

print('Prueba de posesión para text.txt con hash incorrecto:')
if verify_file_cloud(challenge, extension, filename, proof):
    print("Prueba de posesión correcta")
else:
    print("Prueba de posesión ha fallado")
```

En ambos casos, la función se llama con los mismos argumentos, y se imprime un mensaje en la consola para indicar si la prueba de posesión ha sido exitosa o no.
Como podemos comprobar, la primera prueba de posesión tiene un resultado exitoso, mientras que la segunda falla.

```
Prueba de posesión para text.txt con hash correcto:
Prueba de posesión correcta
Prueba de posesión para text.txt con hash incorrecto:
Prueba de posesión ha fallado
```

# Conclusiones

En conclusión, nuestra solución de seguridad ofrece una manera efectiva para que el cliente pueda realizar comprobaciones de sus archivos de sistema y detectar cualquier posible actividad sospechosa. La aplicación desarrollada almacena hashes de archivos en un diccionario y los compara periódicamente para detectar cambios en los archivos. Además, genera informes diarios y mensuales para que el cliente tenga una idea clara de la actividad de sus archivos que ayuda a mantener una visibilidad constante de la actividad en los archivos y a detectar cualquier cambio sospechoso de manera oportuna. También se han implementado pruebas de posesión para nubes no confiables para verificar la autenticidad de los archivos en la nube. En conjunto, todas estas características hacen que nuestra solución sea altamente efectiva en la detección de actividad sospechosa y en la protección de los archivos del cliente.