Informe Proyecto HIDS

Introducción

El Sistema de Detección de Intrusos Basado en el Host (HIDS) es una herramienta diseñada para monitorear y proteger la integridad de archivos en un sistema informático. Este informe proporciona una visión general de las decisiones clave, soluciones implementadas y resultados de pruebas del programa HIDS desarrollado.

Esbozo del proyecto

La idea principal para este HIDS era que tenía que ser escalable y, por lo tanto, las decisiones de Diseño y de Estructura debían ir acordes a estas expectativas, por ello podríamos decir que los 3 pilares de este proyecto informático son los siguientes:

1. **Base de Datos SQLite:** Se eligió utilizar una base de datos SQLite para almacenar los hashes y marcas de tiempo de los archivos. SQLite es una base de datos incorporada en Python que ofrece un rendimiento eficiente y no requiere una configuración compleja.
2. **Hashing SHA-1:** El programa utiliza el algoritmo SHA-1 para calcular los hashes de los archivos. Aunque SHA-1 no es la opción más segura en términos de criptografía, como ya se ha visto en clases de teoría, es adecuado para detectar cambios en los archivos, asumiendo que el atacante no tiene la capacidad de vulnerar este tipo de firma.[1]
3. **Configuración Externa:** Se implementó un archivo de configuración externa llamado "config.ini" para permitir la configuración dinámica de los directorios a monitorear. Esto facilita la adaptabilidad del programa a diferentes entornos.

El programa en sí tiene 3 opciones una vez se ejecuta:

1. **Agregar Directorio:** El programa permite a los usuarios agregar directorios que desean monitorear. Los directorios se almacenan en una lista y se actualizan en el archivo de configuración "config.ini".
2. **Inicializar Base de Datos:** Al seleccionar esta opción, el programa crea una base de datos SQLite y calcula los hashes de todos los archivos en los directorios especificados. Los resultados se almacenan en la base de datos.
3. **Actualizar HIDS:** Esta función compara los hashes almacenados en la base de datos con los hashes recalculados de los archivos en los directorios monitoreados. Si se encuentra una diferencia, se considera una modificación y se registra en un informe mensual. Esta opción también nos sirve para actualizar la propia base de datos

[1] Se ha intentado con el algoritmo SHA-256 y SHA-512, pero los tiempos de espera para hashear tal volumen de archivos se disparaba considerablemente

también si llegase a encontrar un archivo en ese directorio que no estuviera en la base de datos

Problemas y preocupaciones

Una de las principales preocupaciones que he tenido a la hora de diseñar el HIDS es la posibilidad de una modificación indebida del archivo config.ini o de la propia base de datos. He pensado en la posibilidad de encriptar esta base de datos después de cada operación que se hiciera en esta, y en consecuencia se desencriptara también el momento de antes de hacer cualquier operación; aunque debido a un problema en inicio de tipo técnico (todas las implementaciones de encriptado/desencriptado de SQLite que he encontrado están pensadas o exclusivamente, o sin mayores problemas para sistemas UNIX), que se tornó en un problema de falta de tiempo debido a que no he recibido ninguna ayuda de mis compañeros para este proyecto, no he sido capaz de solucionar esta vulnerabilidad.

Igualmente, a modo de "parche" se me ocurrió que también podía añadir a la base de datos una entrada para config.ini y otro para la base de datos, de tal forma que, si no puedo impedir que no modifiquen el archivo, por lo menos, debería de ser capaz de detectar esta modificación; esto sí que está implementado.

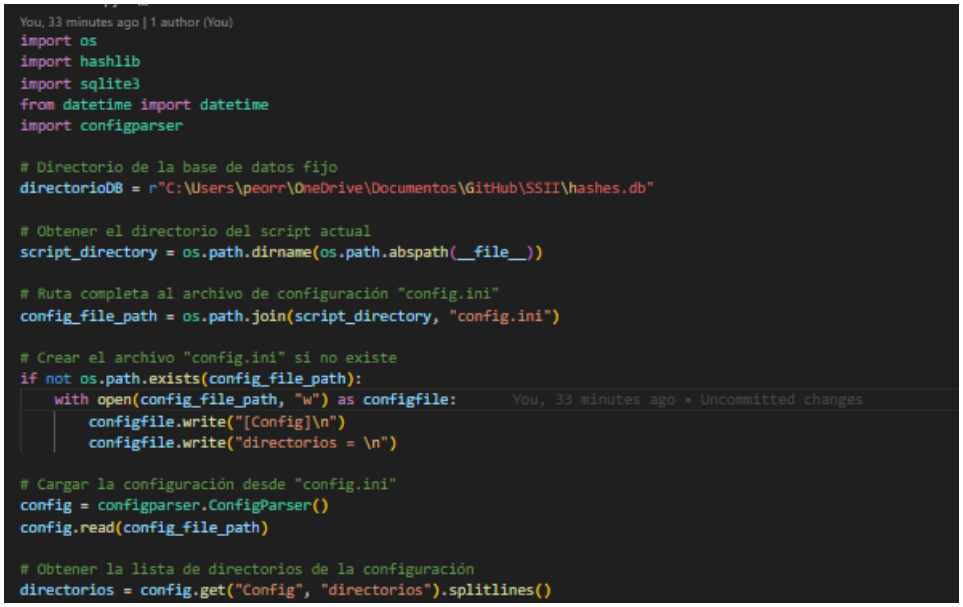
Por otra parte, respecto al objetivo nº 3 "El proceso de verificación se realizará a intervalos, en este caso diariamente y debe almacenarse un informe de un mes entero.", se ha logrado generar un informe que vaya por cada mes, y que en este se registre solamente los archivos modificados indebidamente, pero no se ha conseguido automatizar este proceso, ya que para ello se requieren de permisos especiales que nuevamente son más complicados de conseguir en Windows.

Explicación detallada del programa

Una vez comentado más o menos por encima el proyecto y sus dificultades, voy a explicar las funciones más importantes de este programa:

1. **Configuración y Preparación Inicial:**
    - El programa comienza importando las bibliotecas necesarias y configurando las rutas de archivos y directorios.

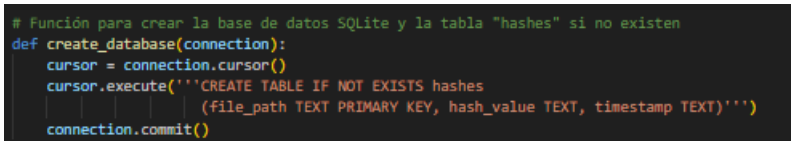
- Se utiliza un archivo de configuración externa llamado "config.ini" para almacenar los directorios a monitorear.



```
You, 33 minutes ago | 1 author (You)
import os
import hashlib
import sqlite3
from datetime import datetime
import configparser

# Directorio de la base de datos fijo
directorioDB = r"C:\Users\peorr\OneDrive\Documentos\GitHub\SSII\hashes.db"

# Obtener el directorio del script actual
script_directory = os.path.dirname(os.path.abspath(__file__))

# Ruta completa al archivo de configuración "config.ini"
config_file_path = os.path.join(script_directory, "config.ini")

# Crear el archivo "config.ini" si no existe
if not os.path.exists(config_file_path):
    with open(config_file_path, "w") as configfile:      You, 33 minutes ago • Uncommitted changes
        configfile.write("[Config]\n")
        configfile.write("directorios = \n")

# Cargar la configuración desde "config.ini"
config = configparser.ConfigParser()
config.read(config_file_path)

# Obtener la lista de directorios de la configuración
directorios = config.get("Config", "directorios").splitlines()
```

2. **Base de Datos SQLite:**

   - Se utiliza una base de datos SQLite para almacenar los hashes y marcas de tiempo de los archivos monitoreados.
   - La función **create_database** crea la tabla "hashes" en la base de datos si no existe.



```
# Función para crear la base de datos SQLite y la tabla "hashes" si no existen
def create_database(connection):
    cursor = connection.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS hashes
                      (file_path TEXT PRIMARY KEY, hash_value TEXT, timestamp TEXT)''')
    connection.commit()
```

3. **Cálculo de Hashes:**

   - La función **hashFile** se utiliza para calcular el hash SHA-1 de un archivo dado.

```
# Función para calcular el hash SHA-1 de un archivo
def hashFile(file_path):
    file_hash = hashlib.sha1()
    with open(file_path, "rb") as f:
        while chunk := f.read(8192):
            file_hash.update(chunk)
    return file_hash.hexdigest()
```

   - Las funciones **hashFiles** y **compareHashes** se encargan de calcular y comparar los hashes de los archivos en los directorios especificados, respectivamente.

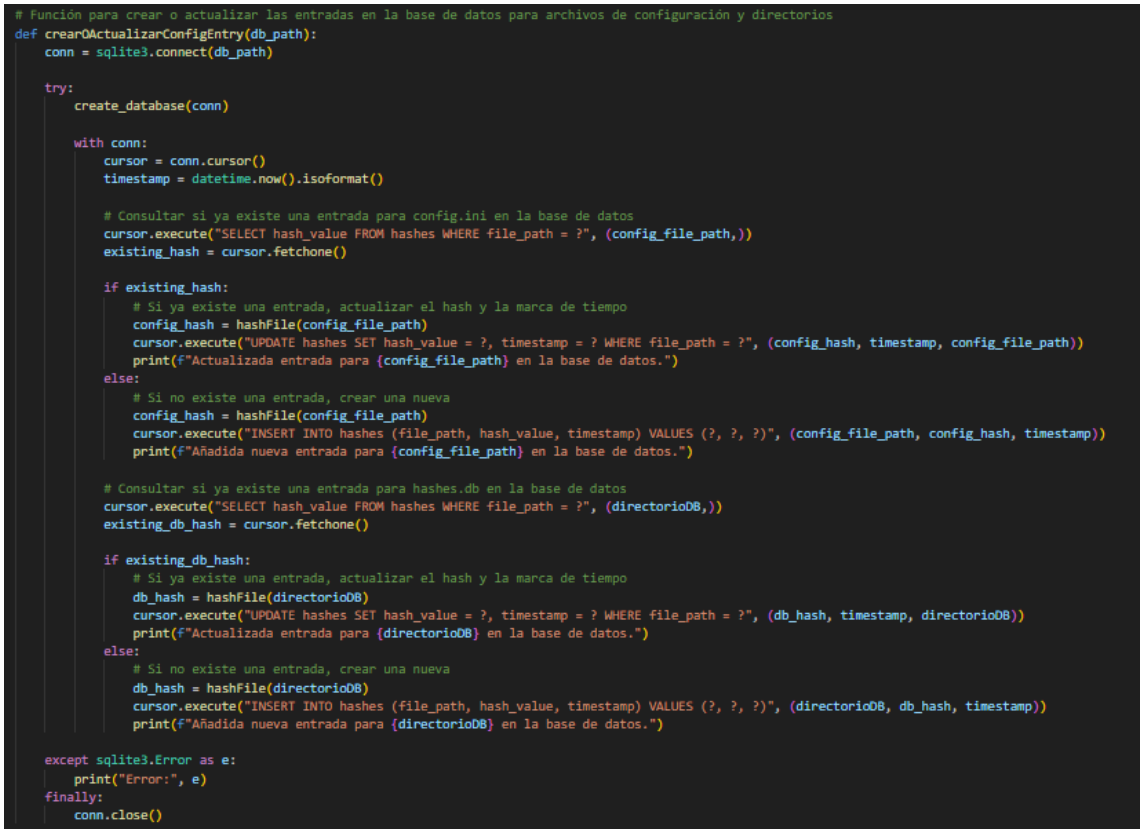
```python
# Función para hashear archivos en el directorio especificado y almacenarlos en la base de datos
def hashFiles(path, db_path):
    file_list = []
    conn = sqlite3.connect(db_path)

    try:
        create_database(conn)

        with conn:
            cursor = conn.cursor()
            for root, dirs, files in os.walk(path):
                for file in files:
                    file_path = os.path.join(root, file)
                    file_list.append(file_path)

                    # Calcular el hash para el archivo actual
                    file_hash = hashFile(file_path)

                    # Obtener la fecha y hora actual en formato ISO
                    timestamp = datetime.now().isoformat()

                    cursor.execute("INSERT OR REPLACE INTO hashes (file_path, hash_value, timestamp) VALUES (?, ?, ?)", (file_path, file_hash, timestamp))

    except sqlite3.Error as e:
        print("Error:", e)
    finally:
        conn.close()

    return file_list
```

```python
# Función para comparar los hashes recalculados de los archivos con los almacenados en la base de datos
def compareHashes(db_path):
    conn = sqlite3.connect(db_path)

    try:
        with conn:
            cursor = conn.cursor()···

                        if file_hash != stored_hash:
                            # Si el hash es diferente, el archivo ha sido modificado
                            print(f"Archivo modificado: {file_path}")
                            print(f"Fecha y hora de creación en la base de datos: {timestamp}")

                            # Obtener la fecha y hora de la última modificación del archivo
                            mod_timestamp = datetime.fromtimestamp(os.path.getmtime(file_path)).isoformat()
                            print(f"Fecha y hora de la última modificación: {mod_timestamp}")

                            # Registrar la modificación en el informe mensual
                            registrarModificacionEnInforme(file_path, mod_timestamp)
                        else:
                            # Si el archivo no está en la base de datos, agregar una nueva entrada
                            file_hash = hashFile(file_path)
                            timestamp = datetime.now().isoformat()
                            cursor.execute("INSERT INTO hashes (file_path, hash_value, timestamp) VALUES (?, ?, ?)", (file_path, file_hash, timestamp))
                            print(f"Añadida nueva entrada para {file_path} en la base de datos.")

    except sqlite3.Error as e:
        print("Error:", e)
    finally:
        conn.close()
```

4. **Gestión de Directorios y Configuración:**

    - El programa permite a los usuarios agregar directorios a monitorear y los almacena en una lista.

- La función **crearOActualizarConfigEntry** [2]actualiza la base de datos con información sobre el archivo de configuración y la base de datos en sí.

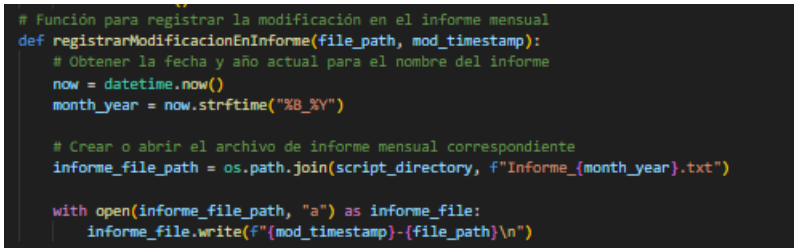


```python
# Función para crear o actualizar las entradas en la base de datos para archivos de configuración y directorios
def crearOActualizarConfigEntry(db_path):
    conn = sqlite3.connect(db_path)

    try:
        create_database(conn)

        with conn:
            cursor = conn.cursor()
            timestamp = datetime.now().isoformat()

            # Consultar si ya existe una entrada para config.ini en la base de datos
            cursor.execute("SELECT hash_value FROM hashes WHERE file_path = ?", (config_file_path,))
            existing_hash = cursor.fetchone()

            if existing_hash:
                # Si ya existe una entrada, actualizar el hash y la marca de tiempo
                config_hash = hashFile(config_file_path)
                cursor.execute("UPDATE hashes SET hash_value = ?, timestamp = ? WHERE file_path = ?", (config_hash, timestamp, config_file_path))
                print(f"Actualizada entrada para {config_file_path} en la base de datos.")
            else:
                # Si no existe una entrada, crear una nueva
                config_hash = hashFile(config_file_path)
                cursor.execute("INSERT INTO hashes (file_path, hash_value, timestamp) VALUES (?, ?, ?)", (config_file_path, config_hash, timestamp))
                print(f"Añadida nueva entrada para {config_file_path} en la base de datos.")

            # Consultar si ya existe una entrada para hashes.db en la base de datos
            cursor.execute("SELECT hash_value FROM hashes WHERE file_path = ?", (directorioDB,))
            existing_db_hash = cursor.fetchone()

            if existing_db_hash:
                # Si ya existe una entrada, actualizar el hash y la marca de tiempo
                db_hash = hashFile(directorioDB)
                cursor.execute("UPDATE hashes SET hash_value = ?, timestamp = ? WHERE file_path = ?", (db_hash, timestamp, directorioDB))
                print(f"Actualizada entrada para {directorioDB} en la base de datos.")
            else:
                # Si no existe una entrada, crear una nueva
                db_hash = hashFile(directorioDB)
                cursor.execute("INSERT INTO hashes (file_path, hash_value, timestamp) VALUES (?, ?, ?)", (directorioDB, db_hash, timestamp))
                print(f"Añadida nueva entrada para {directorioDB} en la base de datos.")

    except sqlite3.Error as e:
        print("Error:", e)
    finally:
        conn.close()
```

- La funcion **agregarDirectorio** nos permite añadir un nuevo directorio al archivo config.ini

5. **Informe de Modificaciones:**

- El programa registra las modificaciones detectadas en un informe mensual. La función **registrarModificacionEnInforme** se encarga de esto.



```python
# Función para registrar la modificación en el informe mensual
def registrarModificacionEnInforme(file_path, mod_timestamp):
    # Obtener la fecha y año actual para el nombre del informe
    now = datetime.now()
    month_year = now.strftime("%B_%Y")

    # Crear o abrir el archivo de informe mensual correspondiente
    informe_file_path = os.path.join(script_directory, f"Informe_{month_year}.txt")

    with open(informe_file_path, "a") as informe_file:
        informe_file.write(f"{mod_timestamp}-{file_path}\n")
```

6. **Menú de Opciones:**

- Se implementa un menú de opciones interactivo que permite a los usuarios realizar las siguientes acciones:
    - Agregar directorios.
    - Inicializar la base de datos y calcular los hashes iniciales.

[2] Esto es lo que se ha comentado anteriormente, gracias a esta función podemos saber si ha sido modificada la db o el archivo config.ini de forma ilícita

- Actualizar el HIDS y detectar modificaciones.
- Salir del programa.

```python
# Función para crear un menú de opciones para el HIDS
def hids_menu():
    while True:
        print("\nMenú de Sistema de Detección de Intrusos Basado en el Host (HIDS):")
        print("1. Agregar Directorio")
        print("2. Inicializar base de datos (Crear la base de datos y hashear archivos)")
            for directorio in directorios:
                print(f"Inicializando HIDS para el directorio: {directorio}")
                file_list = hashFiles(directorio, directorioDB)
                print(f"HIDS inicializado con éxito para el directorio: {directorio}")
        elif choice == '3':
            print("Actualizando HIDS...")
            compareHashes(directorioDB)
            print("HIDS actualizado con éxito.")
        elif choice == '4':
            print("Saliendo del Menú de HIDS.")
            break
        else:
            print("Opción no válida. Por favor, seleccione una opción válida (1, 2, 3 o 4).")

# Ejecutar el menú de HIDS
if __name__ == "__main__":
    hids_menu()
```

Uso del Programa

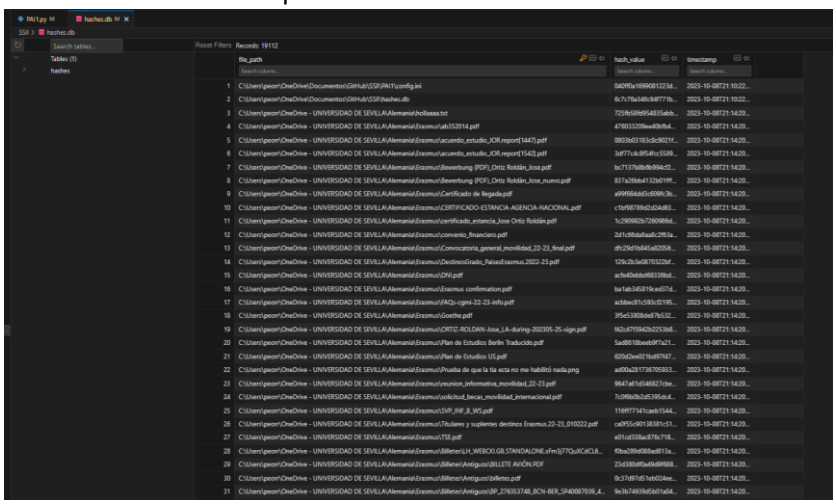
1. Lo primero que haremos será elegir la opción 1, una vez se introduce el directorio, se puede ver que se notifica de la creación de dos nuevas entradas en la db.

```
Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 1
Ingrese el nuevo directorio a analizar (por ejemplo, C:\ruta\al\directorio): C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\A
lemania
Directorio agregado: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania
Añadida nueva entrada para C:\Users\peorr\OneDrive\Documentos\GitHub\SSII\PAI1\config.ini en la base de datos.
Añadida nueva entrada para C:\Users\peorr\OneDrive\Documentos\GitHub\SSII\hashes.db en la base de datos.
```

2. Ahora vamos a inicializar la DB.

```
Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 2
Inicializando HIDS para el directorio: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania
HIDS inicializado con éxito para el directorio: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania
```

   a. Vemos a continuación que en efecto la DB ha sido creada.



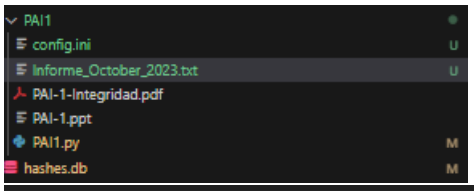
3. Ahora vamos a actualizar el HIDS sin haber hecho ningún cambio.

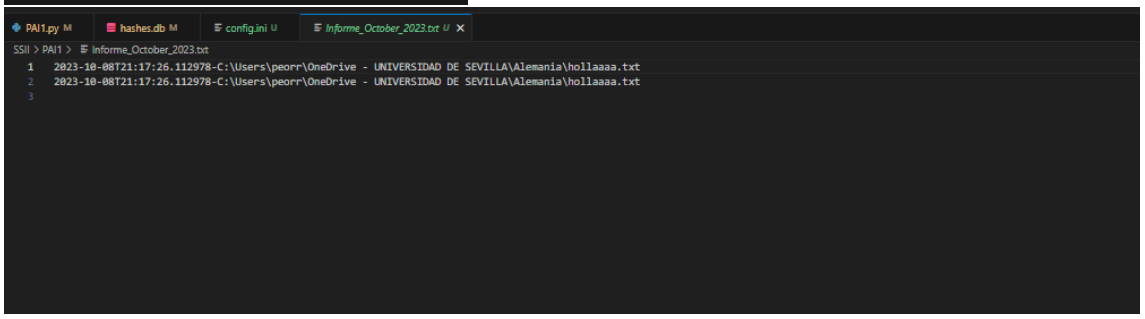
```
Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 3
Actualizando HIDS...
HIDS actualizado con éxito.
```

   a. Vamos a ver qué es lo que pasaría si hacemos un cambio en un archivo.

```
Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 3
Actualizando HIDS...
Archivo modificado: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania\hollaaaa.txt
Fecha y hora de creación en la base de datos: 2023-10-08T21:14:20.579384
Fecha y hora de la última modificación: 2023-10-08T21:17:26.112978
HIDS actualizado con éxito.
```

   Se va a crear entonces el siguiente archivo de Informe para el mes





   Este aviso no se quitará hasta que no volvamos a seleccionar la opción 2[3]:

```
Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 2
Inicializando HIDS para el directorio: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania
HIDS inicializado con éxito para el directorio: C:\Users\peorr\OneDrive - UNIVERSIDAD DE SEVILLA\Alemania

Menú de Sistema de Detección de Intrusos Basado en el Host (HIDS):
1. Agregar Directorio
2. Inicializar base de datos (Crear la base de datos y hashear archivos)
3. Actualizar HIDS (Comparar y actualizar la base de datos)
4. Salir (Salir)
Seleccione una opción: 3
Actualizando HIDS...
HIDS actualizado con éxito.
```

Como se ha visto, si bien no podría considerarse una implementación completa o profesional de un HIDS, es una aproximación bastante fiable al concepto de este, tanto su eficiencia a la hora de buscar (con la ruta de los directorios guardada en config.ini solo se tarda lo que tarde el equipo en leer la información en el disco), calcular (si bien SHA-1 no es seguro a la hora de encriptar archivos, puede sernos útil para este cometido, y de esta forma no sacrificar el

[3] Sé que el nombre de la opción puede inducir a dudas

tiempo de procesamiento) y a la hora de almacenar (usando SQLite podemos escalar cuanto sea necesario esta base de datos y usarla en las comprobaciones)