FILE INTEGRITY MONITOR

Leevi Hokkanen 050253975
Leevi.hokkanen@tuni.fi

Contents

Introduction

The program developed is called *File integrity monitor*. It calculates hashes files and logs them for further observations. These logged hashes can be used for the purpose of comparing them to the files' future hashes.

Comparing hashes can be used for the purpose of determining if these given files have been modified. This method can be used in for example in server databases to determine which files have been changed, verify file versions, or for the purpose of checking for man-in-the-middle attacks.

How use it?

Getting the program to start

The given GitHub repository contains both the source files and a simple executable for the program for Windows. These both can be used to execute the program. The program has only been worked on and tested on Windows 11 operating system, meaning that there is no guarantee whether the program works on other operating systems.

To run the program via the executable file *main.exe* in the repository *main* inside the cloned directory. Simply, start the executable to start the program. **If the program started without issues, you can move on to the next section and ignore the further installations**. **You don't have to read them.**

If the previous alternative didn't work, the program can also be run via the source files, by building a Python Anaconda environment (Anaconda can be downloaded here: https://www.anaconda.com/download). Install Anaconda to your operating system of choice, open the Anaconda command line (or other command line of your choice), and move to the cloned project repository. After this, run the following commands:

```
conda create –n myenv python=3.9

conda activate myenv

pip install eel

python main.py
```

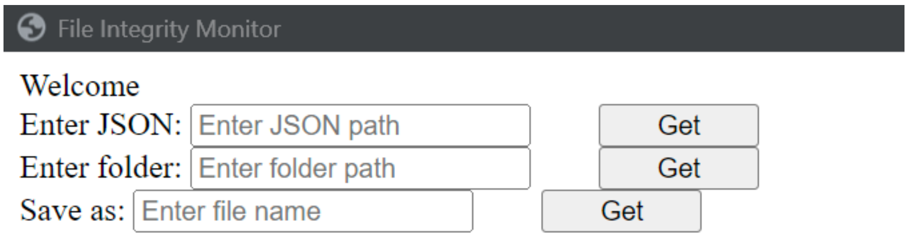
If you receive the *module not found error* for the following commands, try running one of the commands available in the following *StackOverFlow* thread: https://stackoverflow.com/questions/73315265/modulenotfounderror-no-module-named-eel. After this, try running the *python main.py* command again.

There are also screenshots of the program demo if you can't or don't want to install or try the program. If you wish to operate the program and are not able to, try contacting me via the email.

How to quickly demo the program

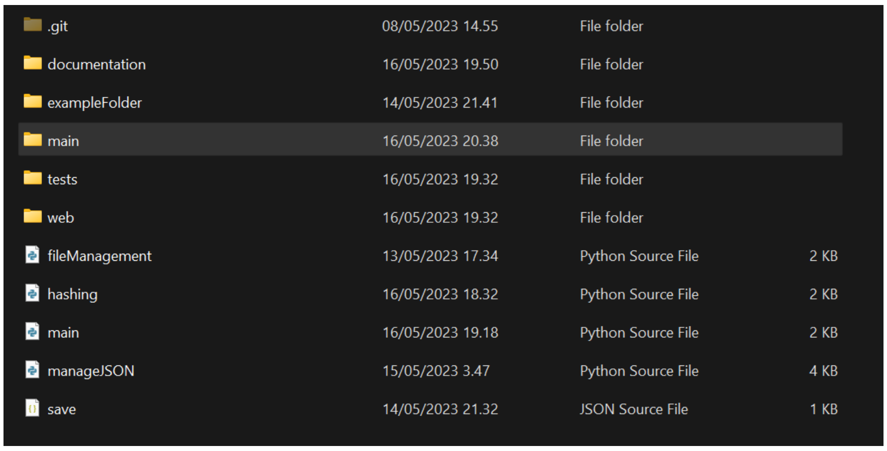
The program allows the user to enter folders to the program, store them into a JSON database, and compare these stored hashes to their new values. The following demo instructs the reader on how to use the program in standard cases.

If you have started the program successfully, you should be greeted with the following window:



Starting window

If you have cloned the entire GitHub directory of the program, it should look something like the following:

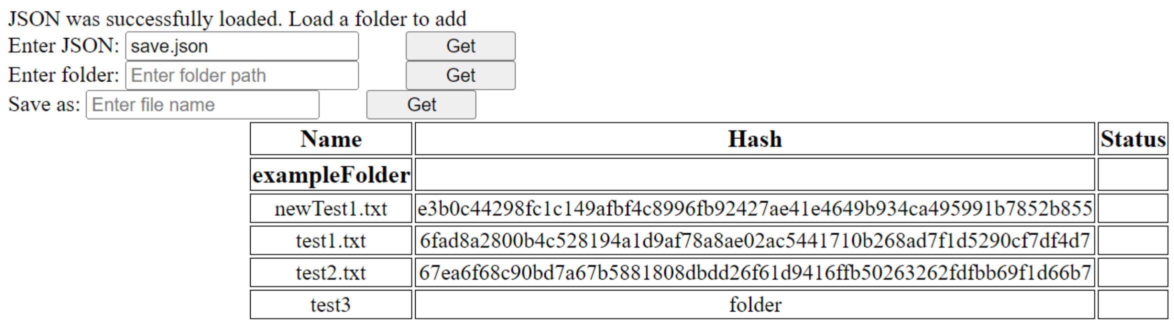


.git	08/05/2023 14.55	File folder	
documentation	16/05/2023 19.50	File folder	
exampleFolder	14/05/2023 21.41	File folder	
main	16/05/2023 20.38	File folder	
tests	16/05/2023 19.32	File folder	
web	16/05/2023 19.32	File folder	
fileManagement	13/05/2023 17.34	Python Source File	2 KB
hashing	16/05/2023 18.32	Python Source File	2 KB
main	16/05/2023 19.18	Python Source File	2 KB
manageJSON	15/05/2023 3.47	Python Source File	4 KB
save	14/05/2023 21.32	JSON Source File	1 KB

What the cloned GitHub directory should look like

In the directory, you have the folder *exampleFolder* which contains a bevy of files which will be used for this demo. There is also database file *save.json*, which contains the hashes of the previous iteration of the previously mentioned *exampleFolder*.

We want to load *save*.json database, to see what has been done the files inside exampleFolder. Copy the path of *save.json* and enter it in to the field *Enter JSON:* and hit *Get*, like in the following picture. You should have a table being built like the following:

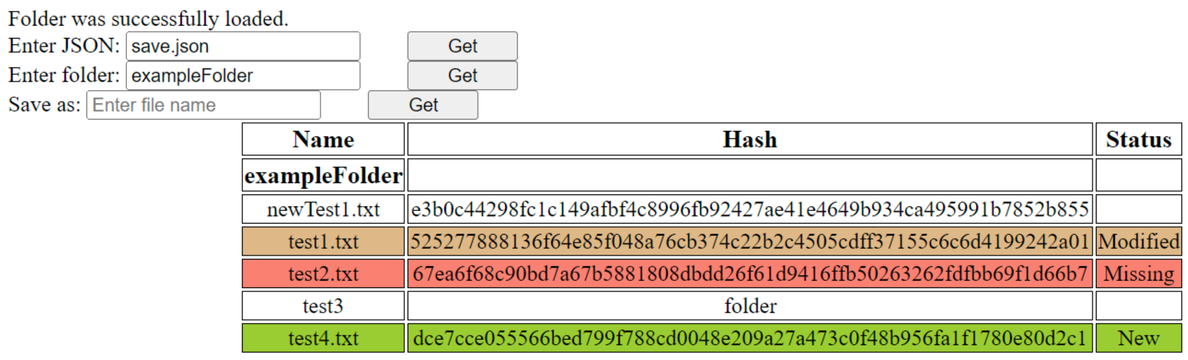

JSON was successfully loaded. Load a folder to add

Enter JSON:	save.json	Get
Enter folder:	Enter folder path	Get
Save as:	Enter file name	Get

Name	Hash	Status
exampleFolder		
newTest1.txt	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855	
test1.txt	6fad8a2800b4c528194a1d9af78a8ae02ac5441710b268ad7f1d5290cf7df4d7	
test2.txt	67ea6f68c90bd7a67b5881808dbdd26f61d9416ffb50263262fdfbb69f1d66b7	
test3	folder	

Loaded table (Write the JSON path to match your installation.)

After this, enter the path of the *exampleFolder* to the *Enter folder:* field and hit *Get*. You should be presented with the following window:


Folder was successfully loaded.

Enter JSON:	save.json	Get
Enter folder:	exampleFolder	Get
Save as:	Enter file name	Get

Name	Hash	Status
exampleFolder		
newTest1.txt	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855	
test1.txt	525277888136f64e85f048a76cb374c22b2c4505cdff37155c6c6d4199242a01	Modified
test2.txt	67ea6f68c90bd7a67b5881808dbdd26f61d9416ffb50263262fdfbb69f1d66b7	Missing
test3	folder	
test4.txt	dce7cce055566bed799f788cd0048e209a27a473c0f48b956fa1f1780e80d2c1	New

Loaded folder (Again, write the folder path to match your installation.)

We can now see how the folder has been modified from its previous iteration. You can update these changes to the *save.json,* by entering its path to the *Save as:* field and hitting *Get*. You can also create another JSON database, by entering the name of a different JSON file to this field.

You can try using these functions to other directories inside your computer. I recommend avoiding folders that have very large files inside them, since it might take a while for those to finish. Avoid also entering anything that might differentiate from standard use to these fields, because the program will break.

How was it developed?

The program was developed using both JavaScript and Python, where the front-end features are built with the former and the backend features on the other hand with the latter.

The program uses Python's *Eel* architecture, which allows the user to build HTML user-interfaces. Python is also used to search for files inside the computer, and also for encryption, and storing of the data.

JavaScript is also used for the purpose of front-end features, but it also takes part in processing of the data.

Criticism

The program is still in pretty rudimentary stage due to a lack of time, however, it has the major key features developed to a point where it can be considered a 1.0 *minimum viable product* version of the program. In this section, I'm going to point out various security and implementation related criticisms of the program that in an ideal situation would have been implemented.

The most glaring security concern is the lack of encryption of the JSON database. The hashes stored inside this database in plaintext, meaning that anyone can enter and modify them to their heart's content. This allows attackers to hide their changes made to the logged files, by simply opening the JSON file and changing the hash to match its current version.

The *script.js* could have split into different files and could have been tested with unit tests.

The user-interface is still rather clunky. The majority of the window is not used, nor does the table scale properly. There has not been any consideration to the colors. Also the user is not able to use the *file explorer* for picking files and directories, which is not ideal.