Cristian Cartofeanu
FAF-121
Information Security
September 28, 2015

Cryptographic hashes

=

```
79054025
255fb1a2
6e4bc422
aef54eb4
```

Objective:

Understand the purpose of hashing algorithms and create a tool that solves a problem using one of them.

Directory files absolute path checker:

This program keeps an eye on the contents of a directory, notifying you when the name of the file has changed. It is ran at regular intervals by a scheduler, comparing the current state of the system with a previous snapshot, reporting differences it found.
Thus, if your system was hacked and a file was renamed you'll know right away.
The software provides the following functionality:

- Show's the number of files in the directory.
- Hashes (using SHA1 algorithm) the absolute path of each file and saves it.
- Every 4 seconds it checks for changes.
- If a change has been detected it shows the original version and the new one of the hash.

Modules I've used:

I used the following NodeJS modules:

fs- stands for FileSystem, it's one of the most important module of NodeJS, All the methods have asynchronous and synchronous forms. In our case we use only asynchronous one.

jshashes- is lightweight library implementing the most extended cryptographic hash function algorithms in pure JavaScript. Supports the following hash algorithms: MD5, SHA1, SHA256, SHA512, HMAC,RIPEMD-160. Additional functionalities: Base64 encoding/decoding and UTF-8 encoding/decoding.

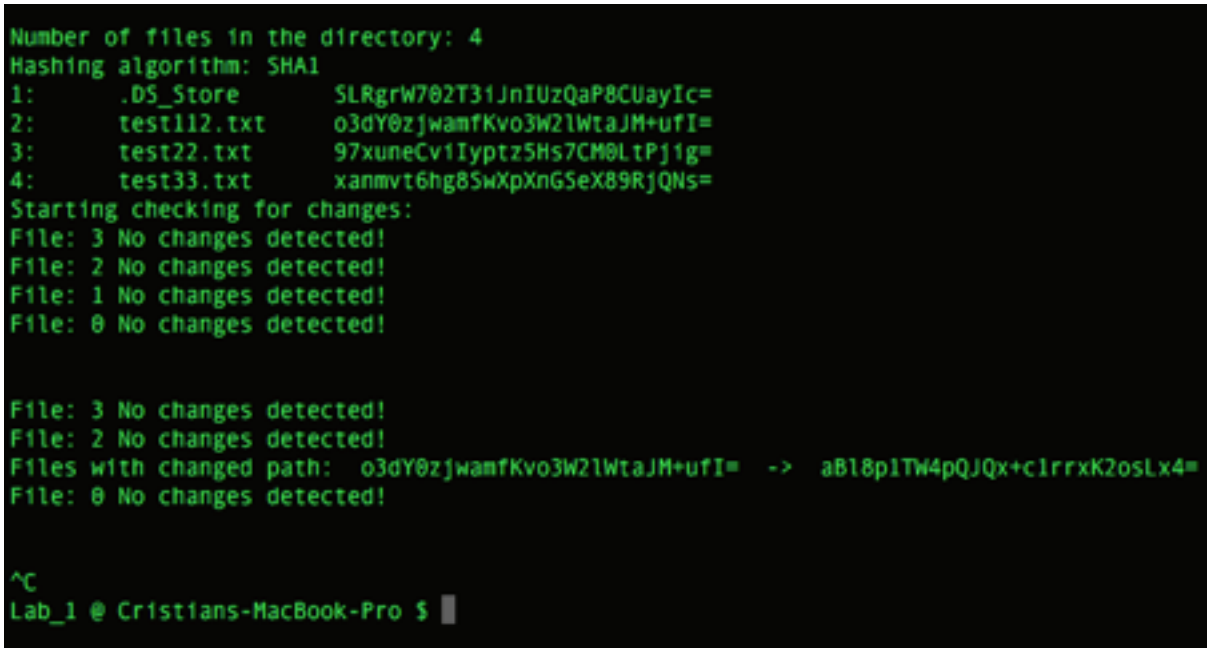
Running our application:

To run our application we need to have already Node.js installed and to download the above modules.

From the terminal we write the following command:

```
node app /path/to/directory/we/want/to/check
```

Expected result:



As we see above the programme shows all the files from the directory we have selected to check and their hashes.

After it starts to verify the integrity of the files absolute path comparing newly generated hash with the original one .

If the program detect a change, it shows the original hash with the new one.

Improving our application

In order to make our application more complex I was working on the idea to check not only the integrity of the file's absolute path but also to check it's content.

Here I use another module for hashing (crypto) in order to show 2 types of implementation that are both almost the same.

I managed to hash one file using (sha512 algorithm) and to output it's hash. When the content of that file changes no mater how much, the hash is also different. This helps us know if files data was modified or not.

When I tried to modify our application from hashing the absolute path of all files into hashing its content I had some errors related with piping the files intro the hash generator that I couldn't manage to solve in a for loop.

Things to be implemented:

- Solving the error with hashing files content in a for loop.
- Better output the difference of witch file was modified.
- To introduce a 3 checking logic
  - If the file was modified
  - If the file was deleted
  - If the file was created

Questions/Answers

1. ***Enumerate several hashing algorithms?***
   MD5, SHA1, SHA256, SHA512,
2. ***What is a collision? Why do collisions exist?***
   Collision in hashing I is when the hash tag of a file is the same with the hash tag of another file.
   Collisions exists because the number of possible output of hashing is smaller then the number of possible inputs.
3. ***When is it a good idea to use MD5?***
   I disagree with the idea that is a good to use MD5 as a hash algorithm.
   The reason is that for MD5 algorithm are already demonstrated some attacks in practice. It's mo secure to use that kind of hashing algorithm that till present are no known successful attacks against this particular algorithm. Some of algorithms with no known successful attacks are SHA256 and SHA521. For the SHA-1 attack is feasible with large amounts of computation power.
4. ***How many bits are there in a SHA1 digest?***
   SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.
5. ***Which hashing algorithm is the fastest? Why? How do you know?***
   The fastest hashing algorithm could be CRC32. But we should be aware that CRC32 will have more collisions than MD5 or even SHA-1 hashes, simply because of the reduced length (32 bits compared to 128 bits respectively 160 bits). But if we just want to check whether a stored string is corrupted, you'll be fine with CRC32. I found about this on the stack overflow .

6. If files are larger than the size of your RAM?

My fstream is a stream reader, it will read file per small blocks of fixed size, and our hash will be building sha512 hash string off small blocks of the file. Our program does not hold the entire file in ram, it reads small chunks of it, generate hash digest, evict them from ram and read the next chunks of data. If we want to test this, we can generate our 11GB file fallocate -l 11G 10gbtest.img and run our script against this file.

Conclusion:

Hashes are a bit like fingerprints for data.
This paper has presented some basic ideas, algorithms and criteria for producing an application that hashes files absolute path and it's content.

Git

https://github.com/CristianChris/UTM-SI