System Integrity Verifier (SIV)

Name: Md Rabiul Ahamed Bin Hanif
Acronym: mdhn18
Personal Number: 19920212-2637
Email: mdhn18@student.bth.se
Course: Network Security, ET2595

Introduction:

The **aim** of this SIV (System Integrity Verifier) assignment is to detect the file **system** modifications occurring within a directory tree for a Linux **system**. The outputs of this assignment statistics and warnings about changes of file modifications to a report file. Thereby, SIV verifies the **integrity** of a monitored directory.

- It can be detecting the modification of file and directory.
- Detected the harmful file which is made by hacker.
- It's able to give warning notification.
- Specify the modification file name.
- Provide extra protection.

Design and Implementation:

1. Read the verification file and decode JSON data.

2. Loop for recursively iterate through the monitored directory contents.

a.
Count number of directories and file.

b.
Find the path or calculate full path, file size, user name, group name, access right, last modification date, file hash content (only for file) in verification mode.

c.
Compare between initialization stage generated data in verification mode for each directory/ file content.

d.
If directory/ file path is found in previous stored data:
that operation will execute which is given below

➔ **File**
If previous stored file size and new generated file size is not same:
warning count and print warning message.

➔ **User**
If previous stored user name and new generated user is not same:
warning count and print warning message.

➔ **Group**
If previous stored group name and new generated group name is not same:
warning count and print warning message.

➔ **Access**
If previous stored access right and new generated access is not same:
warning count and print warning message.

e. else:
New Directory/ File:
If directory/ file path is not found in previous stored data:
warning count and print warning message.

3. Remove Directory/ File:
If directory/ file path is not found in new generated data:
warning count and print warning message.

4. I there is invalid something the system will exit. Need to try newly with proper corrections.

Verification file format:
in this section, verification file format is JSON. Here three elements

i. directory
ii. file
iii. hash function

For both directory and file the following information has been stored to compare information in the verification mode.

- File location/directory
- Accessibility
- Group
- User
- File size
- Last modifications
- Hash

Example:
For directory:
```
"/home/samir/Downloads/rabiul/siv (1)": {
        "Accessibility": "0o775",
            "Group": "samir",
            "Last Modification Time": "Fri Dec 29 12:05:46 2017",
            "Size": 4096,
            "User": "samir"
        }
    },
For file:
    {
        "/home/samir/Downloads/rabiul/1819944775_1604924908_1872217121134958162(1).docx": {
            "Accessibility": "0o664",
            "Group": "samir",
            "Last Modification Time": "Tue Nov  3 07:28:49 2020",
            "Size": 568058,
            "User": "samir",
            "hash_type": "03d7936440da6f1c94d100e4967d8cb091999452"
   },
For hash:
{
        "hash_type": "sha1"
    }
```

Initialization mode report Format:

```
              ***********************************
              # # # Initialization mode report # # #
              ***********************************

         Monitored directory :  orig/data

  Verification file location :  orig/vDB

        Report file location :  orig/init.txt

Number of directories parsed :  2

      Number of files parsed :  3

                  Total Time :  0:00:00.001382
```

Verification mode report format:

```
              *********************************
              # # # Verification mode report # # #
              *********************************
```

```
       Monitored directory :  orig/data
  Verification file location :  orig/vDB
       Report file location :  orig/verify.txt
Number of directories parsed :  1
      Number of files parsed :  2
             Total Time  :  0:00:00.001519
         Number of Warnings :  6
```

Language:
I did this SIV by Python programming language.

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++. The Python is widely used in bigger organizations because of its multiple programming paradigms.

The python language is one of the most accessible programming languages available because it has simplified syntax and not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

These are the reasons for choosing Python programming language for SIV assignment.

Software Dependencies

Usage:
Run help from Ubuntu command line.

```
python siv.py -h
```

Run initialization mode from Ubuntu command line.

```
python3 siv.py -i -D /monitored_directory-V verification_file.txt -R report1.txt -H SHA1
```

Example for run my code initialization mode :

```
python3 siv.py -i -D /home/samir/Downloads/rabiul/ -V /home/samir/Documents/SIV_files/v_file.txt -
R /home/samir/Documents/SIV_files/ri_file.txt -H SHA-1
```

Run verification mode from Ubuntu command line

```
python3 siv.py -v -D /monitored_directory-V verification_file.txt -R report2.txt -H SHA1
```

Example for run my code verification mode:

```
python3 siv.py -v -D /home/samir/Downloads/rabiul/ -V /home/samir/Documents/SIV_files/v_file.txt -
R /home/samir/Documents/SIV_files/rv_file.txt
```

Run this code:

I run this code by test framework, provided by a GitHub link recently in the announcements section. This code able to run by the test framework.

Command line:

```
./test_siv.py -s ./siv.py -e orig
```

Limitations

This assignment can't not rum below python 3 version.

Pseudo-code:

- In help mode, the program will show usage and then terminates.

```
if sys.argv[1] == "-h":
```

- Here is the argument for help

```
parser = argparse.ArgumentParser()
arg_group = parser.add_mutually_exclusive_group()
arg_group.add_argument("-i", "--initialize", action="store_true", help="Initialization mode")
arg_group.add_argument("-v", "--verify", action="store_true", help="Verification mode")
parser.add_argument("-D", "--monitored_directory", type=str, help="Write the name of the directory
that you want to monitor")
parser.add_argument("-V", "--verification_file", type=str,help="Write the Verification File path that
can store information of directories & files in the monitored directory")
parser.add_argument("-R", "--report_file", type=str, help="Write the name of the Report File to store
Initialization / Verification report")
parser.add_argument("-H", "--hash_function", type=str, help="Write name of the hash function,
supported hashes are 'SHA-1' and 'MD-5'")

args = parser.parse_args()
```

1. Defining parameters to simplify

- Monitored directory

```
mon_dir = args.monitored_directory
```

- Verification file path

```
verify_file_path = args.verification_file
```

- Report file path

```
report_file_path = args.report_file
```

- Hash function

```
hash = args.hash_function
```

2. Check if specified monitored directory exist or not

```
def check_mon_dir_exist():
```

3. his function will check if verification file is in the monitored directory

```
def check_verif_in_mon_dir():
```

4. This function will check if report file is in the monitored directory

```
def check_report_in_mon_dir():
```

5. This function will check if the hash function is supported

```
def check_hash_supported():
```

6. This function will check if the verification file already exists

```
def verification_file_exist():
```

7. This function will check if the report file already exists

```
def report_file_exist():
```

8. If the verification file or report file exists already then, the user will be asked if he wants to overwrite the existing file. If answer is “no”, the program terminates.

```
if verification_file_exist():
variable_to_check_overwrite= input("...? y/n: ")
if variable_to_check_overwrite== "y":/
/recursively iterate through the directory contents
//writeto the verification file
//write to the reportfile
```

9. write into verification file

```
 data.append(data_dir)
 data.append(data_file)
 data.append(data_hash)
 json_string = json.dumps(data, indent=4, sort_keys=True)
```

10. verification file structure (JSON data)

```
[
{
  "/d_path": {
          "d_access": "...",
          "d_group": "..."
          "d_date": "...",
          "d_size": ...,
          "d_user": "…"
          },
…

},
{"/f_path": {
          "f_access": "...",
          "f_group": "...",
          "f_date": "...",
          "f_size": ...,
          "f_user": "…"
          "f_hash": "...",
   },
   {
      "hash_type": "sha1"
   }

]
```

11. Initialization mode report file structure (JSON data)

Full pathname to monitored directory: /monitored_d
Full pathname to verification file: os.path.abspath(v_file_path)
Number of directories parsed: total_directories
Number of files parsed: total_files
Total time taken: total_time

12. Verification report file structure (JSON data)

Full pathname to monitored directory: /monitored_d
Full pathname to verification file: os.path.abspath(v_file_path)
Number of directories parsed: total_directories
Number of directories parsed: total_files
Total time taken: total_files
Number of warnings: