System Integrity Verifier (SIV)

1 INTRODUCTION

This simple *System Integrity Verifier (SIV)* has been developed in *Python*. It is capable of detecting any changes or modifications, removal or addition of files or directories in a *UNIX filesystem*. In short, the fundamental goal of the assignment is to learn how to secure filesystem of the *UNIX based system* from the intruder or unknown user who are not authorized to use and modify anything in the filesystem and also to identify any kind of changes, removals or additions occurring within the specified directory tree of *UNIX based system*. *SIV* application helps to secure the filesystem and data by providing Integrity-Information inside the filesystem.

2 DESIGN AND IMPLEMENTATION

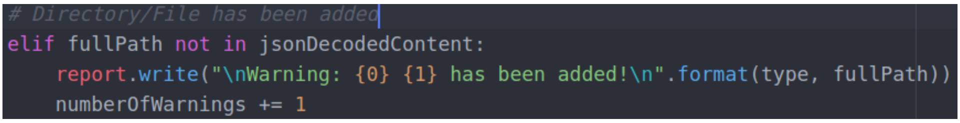
2.1 Demonstration of Six Changes

2.1.1 New or removed files/directories

2.1.1.1 File/Directory Added

If any ***RecursivelyWalkedRecord*** (like - directory / file) of the monitored directory is not found in the ***VerificationFileRecord***, then the record is assumed to be added newly.

```
if RecursivelyWalkedRecord not in VerificationFileRecord:
        report.write(“Warning: File/Directory has been added!”)
```



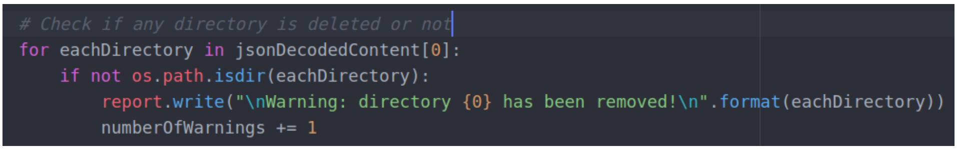
```python
# Directory/File has been added
elif fullPath not in jsonDecodedContent:
    report.write("\nWarning: {0} {1} has been added!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check directory or file addition.

2.1.1.2 File/Directory Removed

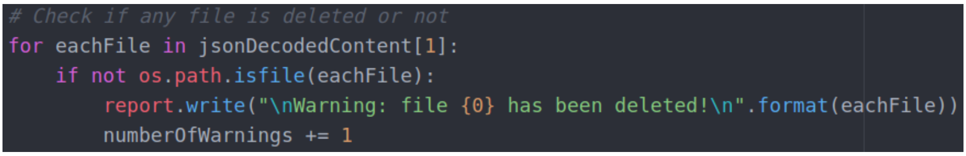
If any ***VerificationFileRecord*** (like - directory / file) fails to certify that it is a valid directory / file, then the record is assumed to be removed.

```
if VerificationFileRecord is not ValidRecord:
        report.write(“Warning: File/Directory has been removed!”)
```



```python
# Check if any directory is deleted or not
for eachDirectory in jsonDecodedContent[0]:
    if not os.path.isdir(eachDirectory):
        report.write("\nWarning: directory {0} has been removed!\n".format(eachDirectory))
        numberOfWarnings += 1
```

Figure: Check directory removal.



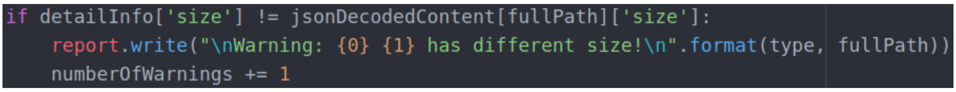
```python
# Check if any file is deleted or not
for eachFile in jsonDecodedContent[1]:
    if not os.path.isfile(eachFile):
        report.write("\nWarning: file {0} has been deleted!\n".format(eachFile))
        numberOfWarnings += 1
```

Figure: Check file removal.

2.1.2 Files with a different size than recorded

If ***RecursivelyWalkedRecord****'s* ***CurrentSize*** does not match with ***VerificationFileRecord****'s* ***SavedSize***, then the record is assumed to be a different size than recorded before.

```
if CurrentSize not equal to SavedSize:
        report.write("Warning: File/Directory has a different size than recorded!")
```



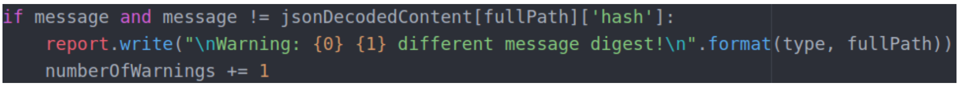
```
if detailInfo['size'] != jsonDecodedContent[fullPath]['size']:
    report.write("\nWarning: {0} {1} has different size!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in file size.

2.1.3 Files with a different message digest than computed before

If ***RecursivelyWalkedRecord****'s* ***CurrentMessageDigest*** does not match with ***VerificationFileRecord****'s* ***SavedMessageDigest***, then the record is assumed to be a different message digest than computed before.

```
if CurrentMessageDigest not equal to SavedMessageDigest:
        report.write("Warning: File with a different message digest!")
```



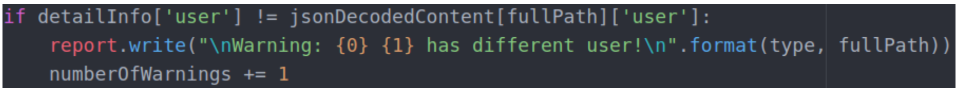
```
if message and message != jsonDecodedContent[fullPath]['hash']:
    report.write("\nWarning: {0} {1} different message digest!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in message digest.

2.1.4 Files/directories with a different user/group

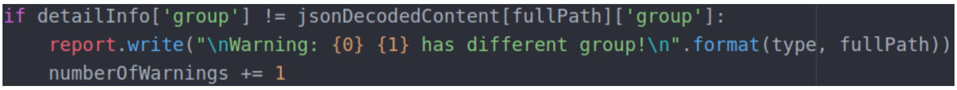
If ***RecursivelyWalkedRecord****'s* ***CurrentUser/Group*** does not match with ***VerificationFileRecord****'s* ***SavedUser/Group***, then the record is assumed to be with a different user/group.

```
if CurrentUser/Group not equal to SavedUser/Group:
        report.write("Warning: Files/directories with a different user/group!")
```



```
if detailInfo['user'] != jsonDecodedContent[fullPath]['user']:
    report.write("\nWarning: {0} {1} has different user!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in user.



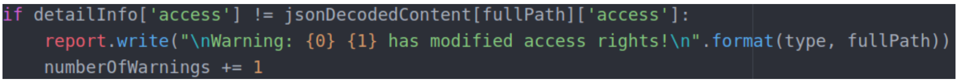
```
if detailInfo['group'] != jsonDecodedContent[fullPath]['group']:
    report.write("\nWarning: {0} {1} has different group!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in group.

2.1.5 Files/directories with modified access right

If ***RecursivelyWalkedRecord****'s* ***CurrentAccessPermission*** does not match with ***VerificationFileRecord****'s* ***SavedAccessPermission***, then the record is assumed to be with modified access right.

```
if CurrentAccessPermission not equal to SavedAccessPermission:
        report.write("Warning: Files/directories with modified access right!")
```



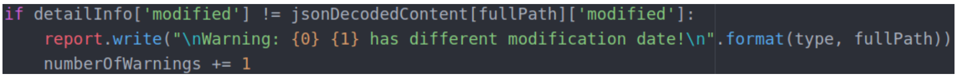
```
if detailInfo['access'] != jsonDecodedContent[fullPath]['access']:
    report.write("\nWarning: {0} {1} has modified access rights!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in access right.

2.1.6 Files/directories with a different modification date

If ***RecursivelyWalkedRecord****'s* ***CurrentModificationDate*** does not match with ***VerificationFileRecord****'s* ***SavedModificationDate***, then the record is assumed to be with a different modification date.

```
if CurrentModificationDate does not match with SavedModificationDate:
        report.write("Warning: Files/directories with a different modification date!")
```



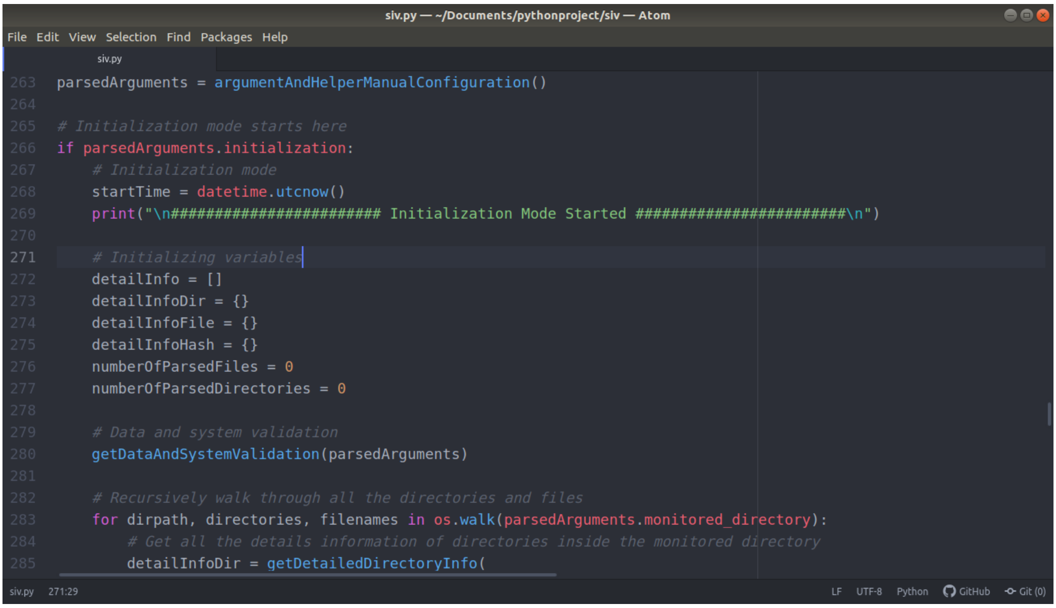
```
if detailInfo['modified'] != jsonDecodedContent[fullPath]['modified']:
    report.write("\nWarning: {0} {1} has different modification date!\n".format(type, fullPath))
    numberOfWarnings += 1
```

Figure: Check difference in access right.

2.2 Description of Algorithm

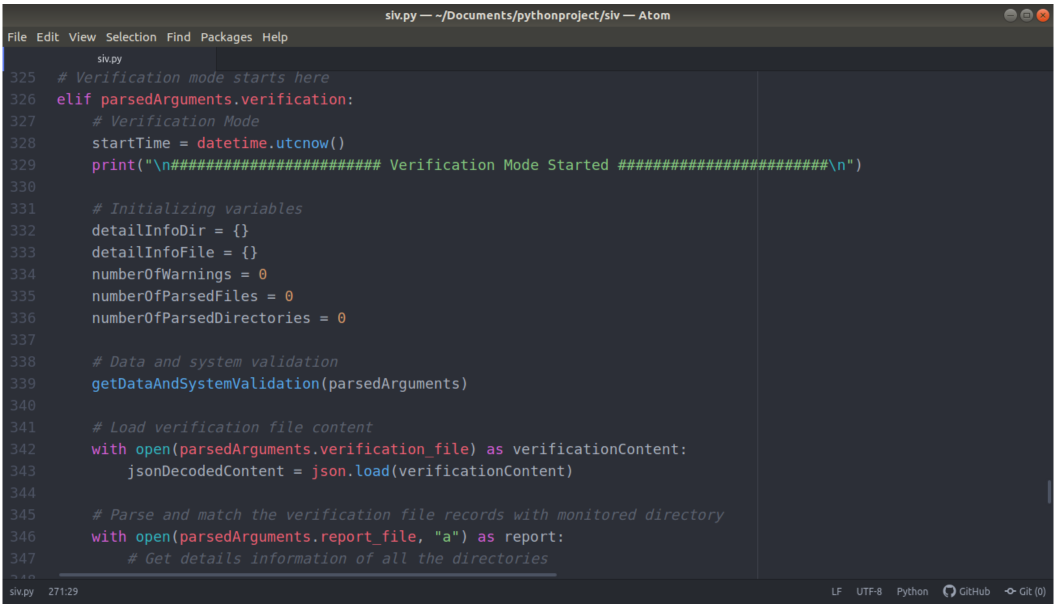
Basically, rather than using any high level algorithm, I used general logical process to develop this simple *System Integrity Verifier (SIV)* application. I tried to keep the code as simple as possible and tried to write function/method based script, where I tried to ensure the reuse of functions/methods for the almost same purpose. I developed functions/methods for serving specific purposes.

At first, I setup all the necessary arguments for the application commands and built the helper manual for the usage of *SIV* application. Then I separated the code for two modes, i.e. *Initialization* and *Verification*.



```
parsedArguments = argumentAndHelperManualConfiguration()

# Initialization mode starts here
if parsedArguments.initialization:
    # Initialization mode
    startTime = datetime.utcnow()
    print("\n######################## Initialization Mode Started ########################\n")

    # Initializing variables
    detailInfo = []
    detailInfoDir = {}
    detailInfoFile = {}
    detailInfoHash = {}
    numberOfParsedFiles = 0
    numberOfParsedDirectories = 0

    # Data and system validation
    getDataAndSystemValidation(parsedArguments)

    # Recursively walk through all the directories and files
    for dirpath, directories, filenames in os.walk(parsedArguments.monitored_directory):
        # Get all the details information of directories inside the monitored directory
        detailInfoDir = getDetailedDirectoryInfo(
```

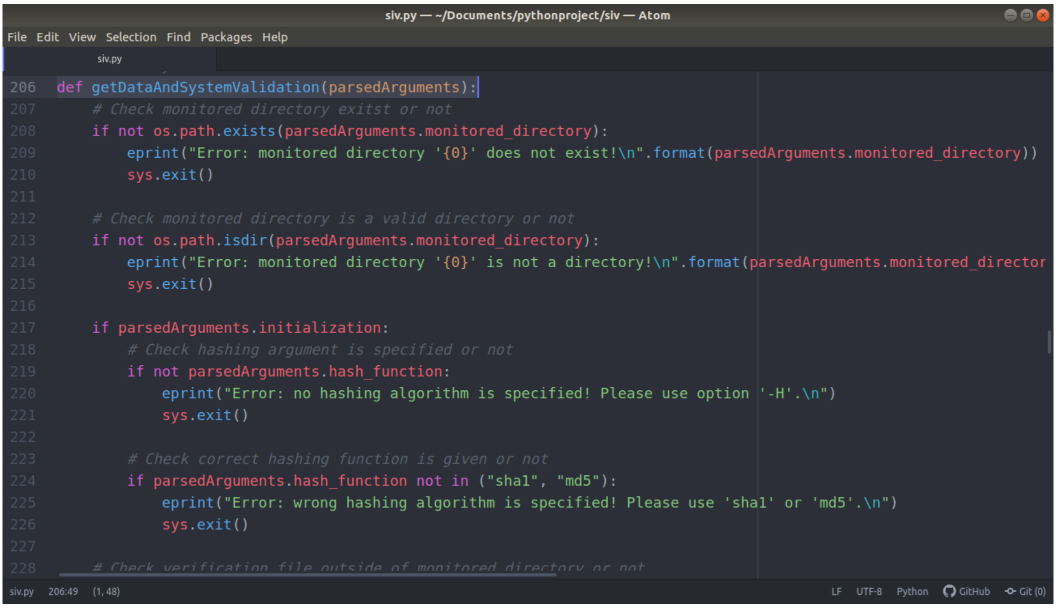
Figure: Initialization mode.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 271:29
LF UTF-8 Python GitHub Git (0)

```python
# Verification mode starts here
elif parsedArguments.verification:
    # Verification Mode
    startTime = datetime.utcnow()
    print("\n######################## Verification Mode Started ########################\n")

    # Initializing variables
    detailInfoDir = {}
    detailInfoFile = {}
    numberOfWarnings = 0
    numberOfParsedFiles = 0
    numberOfParsedDirectories = 0

    # Data and system validation
    getDataAndSystemValidation(parsedArguments)

    # Load verification file content
    with open(parsedArguments.verification_file) as verificationContent:
        jsonDecodedContent = json.load(verificationContent)

    # Parse and match the verification file records with monitored directory
    with open(parsedArguments.report_file, "a") as report:
        # Get details information of all the directories
```

Figure: Verification mode.

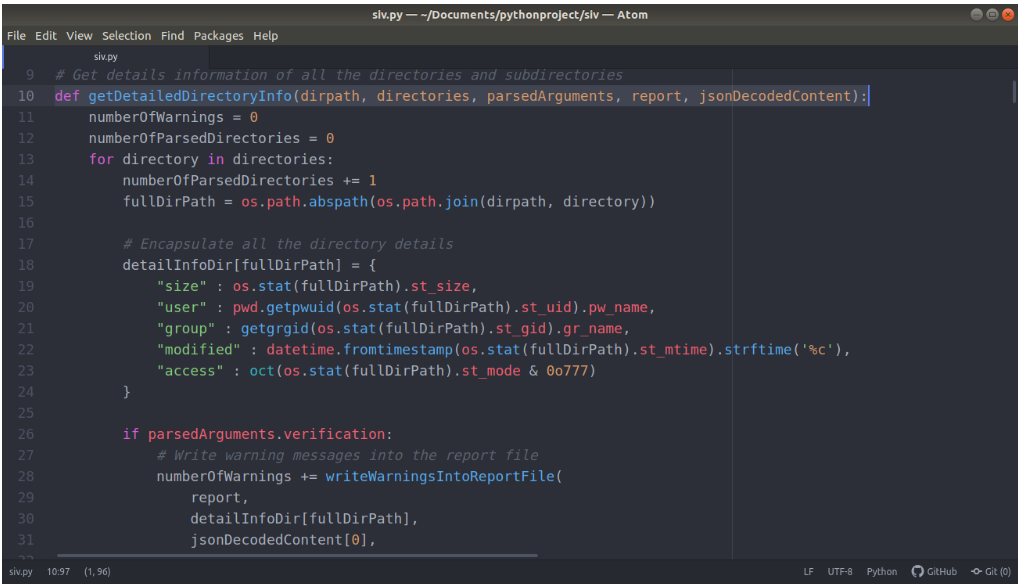
I developed a simple data validation method/function, ***getDataAndSystemValidation()***, for validating all the required data validation constraints specified in the *SIV* application manual. The validation method is responsible for validating all the data constraints for both *Initialization* and *Verification* modes of the application.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 206:49 (1, 48)
LF UTF-8 Python GitHub Git (0)

```python
def getDataAndSystemValidation(parsedArguments):
    # Check monitored directory exitst or not
    if not os.path.exists(parsedArguments.monitored_directory):
        eprint("Error: monitored directory '{0}' does not exist!\n".format(parsedArguments.monitored_directory))
        sys.exit()

    # Check monitored directory is a valid directory or not
    if not os.path.isdir(parsedArguments.monitored_directory):
        eprint("Error: monitored directory '{0}' is not a directory!\n".format(parsedArguments.monitored_director
        sys.exit()

    if parsedArguments.initialization:
        # Check hashing argument is specified or not
        if not parsedArguments.hash_function:
            eprint("Error: no hashing algorithm is specified! Please use option '-H'.\n")
            sys.exit()

        # Check correct hashing function is given or not
        if parsedArguments.hash_function not in ("sha1", "md5"):
            eprint("Error: wrong hashing algorithm is specified! Please use 'sha1' or 'md5'.\n")
            sys.exit()

    # Check verification file outside of monitored directory or not
```

Figure: Data and system constraints validation method.

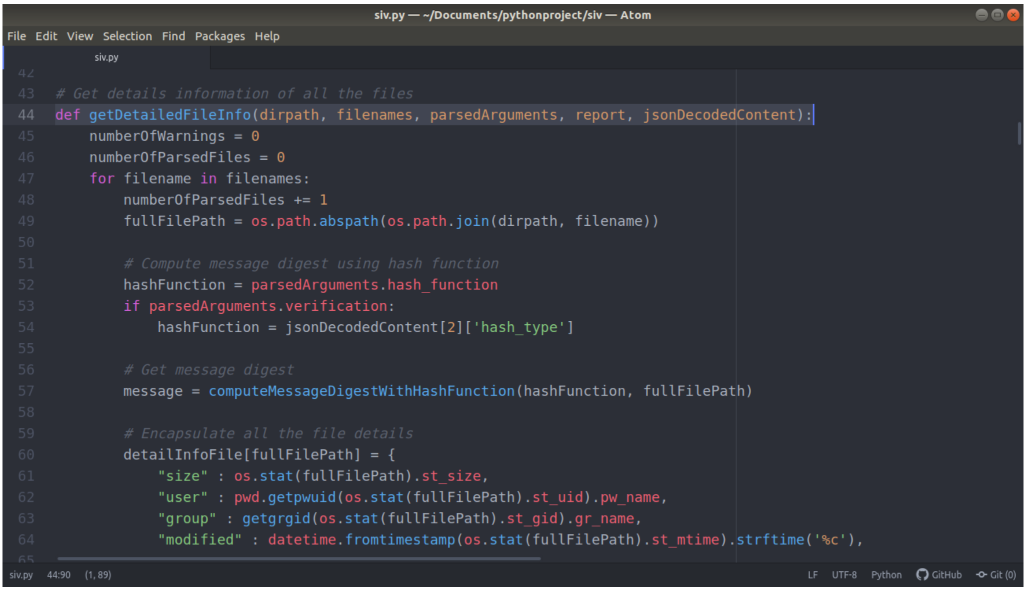
After validating all the necessary data and constraints, I recursively walked through all the directories, subdirectories and then files of the specified monitored directory. Initially I recursively walked through all the directories and subdirectories of the specified monitored directory. For serving that purpose, I developed a comprehensive method, ***getDetailedDirectoryInfo()***, which works for both *Initialization* and *Verification* modes and returns all the details of the directories or subdirectories.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 10:97 (1, 96)
LF UTF-8 Python GitHub Git (0)

```python
# Get details information of all the directories and subdirectories
def getDetailedDirectoryInfo(dirpath, directories, parsedArguments, report, jsonDecodedContent):
    numberOfWarnings = 0
    numberOfParsedDirectories = 0
    for directory in directories:
        numberOfParsedDirectories += 1
        fullDirPath = os.path.abspath(os.path.join(dirpath, directory))

        # Encapsulate all the directory details
        detailInfoDir[fullDirPath] = {
            "size" : os.stat(fullDirPath).st_size,
            "user" : pwd.getpwuid(os.stat(fullDirPath).st_uid).pw_name,
            "group" : getgrgid(os.stat(fullDirPath).st_gid).gr_name,
            "modified" : datetime.fromtimestamp(os.stat(fullDirPath).st_mtime).strftime('%c'),
            "access" : oct(os.stat(fullDirPath).st_mode & 0o777)
        }

        if parsedArguments.verification:
            # Write warning messages into the report file
            numberOfWarnings += writeWarningsIntoReportFile(
                report,
                detailInfoDir[fullDirPath],
                jsonDecodedContent[0],
```

Figure: Method that provides detailed information of directories and subdirectories.

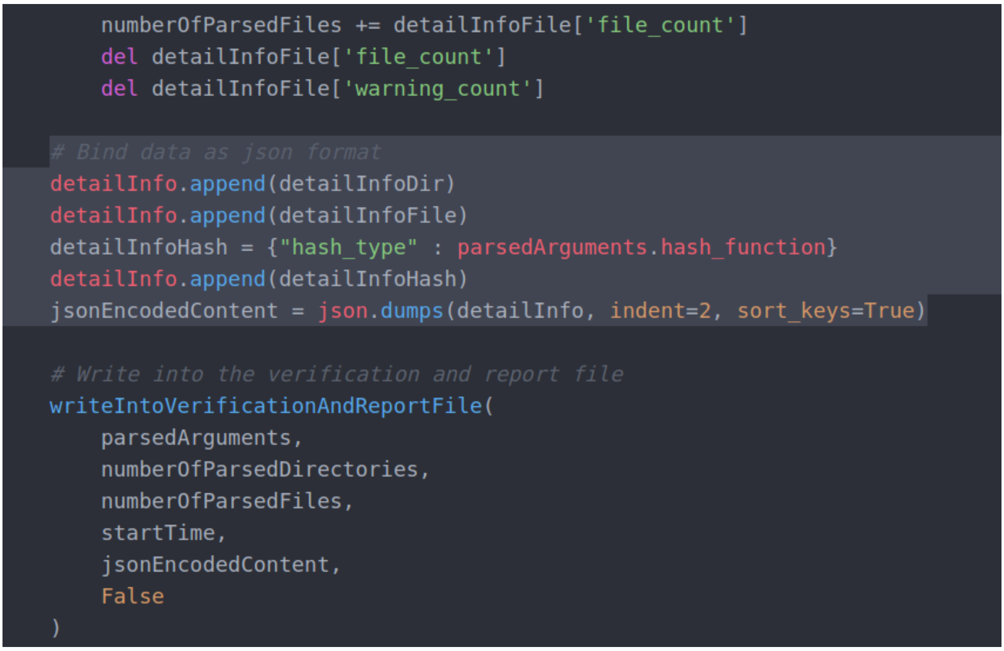
After getting all the directory details, I recursively walked through all the files of the specified monitored directory. For serving that purpose, I developed another comprehensive method, ***getDetailedFileInfo()***, which works for both *Initialization* and *Verification* modes and returns all the details of the files in the monitored directory.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 44:90 (1, 89)
LF UTF-8 Python GitHub Git (0)

```python
# Get details information of all the files
def getDetailedFileInfo(dirpath, filenames, parsedArguments, report, jsonDecodedContent):
    numberOfWarnings = 0
    numberOfParsedFiles = 0
    for filename in filenames:
        numberOfParsedFiles += 1
        fullFilePath = os.path.abspath(os.path.join(dirpath, filename))

        # Compute message digest using hash function
        hashFunction = parsedArguments.hash_function
        if parsedArguments.verification:
            hashFunction = jsonDecodedContent[2]['hash_type']

        # Get message digest
        message = computeMessageDigestWithHashFunction(hashFunction, fullFilePath)

        # Encapsulate all the file details
        detailInfoFile[fullFilePath] = {
            "size" : os.stat(fullFilePath).st_size,
            "user" : pwd.getpwuid(os.stat(fullFilePath).st_uid).pw_name,
            "group" : getgrgid(os.stat(fullFilePath).st_gid).gr_name,
            "modified" : datetime.fromtimestamp(os.stat(fullFilePath).st_mtime).strftime('%c'),
```

Figure: Method that provides detailed information of files.

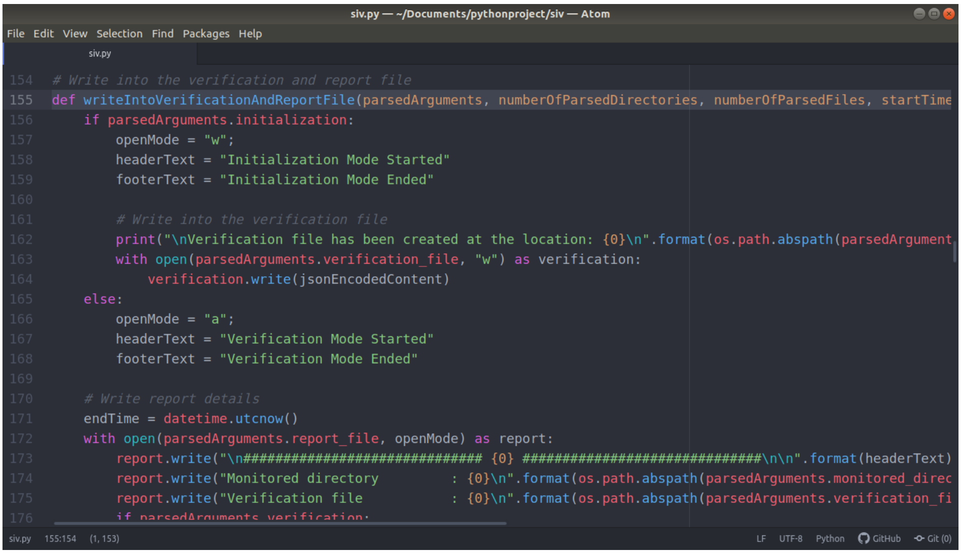
And next, I encapsulate all the directory information and file information that got from ***getDetailedDirectoryInfo()*** and ***getDetailedFileInfo()*** functions successively. After encapsulating the data, I dump that *JSON formatted data* into *JSON Object* (only for Initialization mode).



```
        numberOfParsedFiles += detailInfoFile['file_count']
        del detailInfoFile['file_count']
        del detailInfoFile['warning_count']

    # Bind data as json format
    detailInfo.append(detailInfoDir)
    detailInfo.append(detailInfoFile)
    detailInfoHash = {"hash_type" : parsedArguments.hash_function}
    detailInfo.append(detailInfoHash)
    jsonEncodedContent = json.dumps(detailInfo, indent=2, sort_keys=True)

    # Write into the verification and report file
    writeIntoVerificationAndReportFile(
        parsedArguments,
        numberOfParsedDirectories,
        numberOfParsedFiles,
        startTime,
        jsonEncodedContent,
        False
    )
```

Figure: Dumping JSON formatted data into JSON Object.

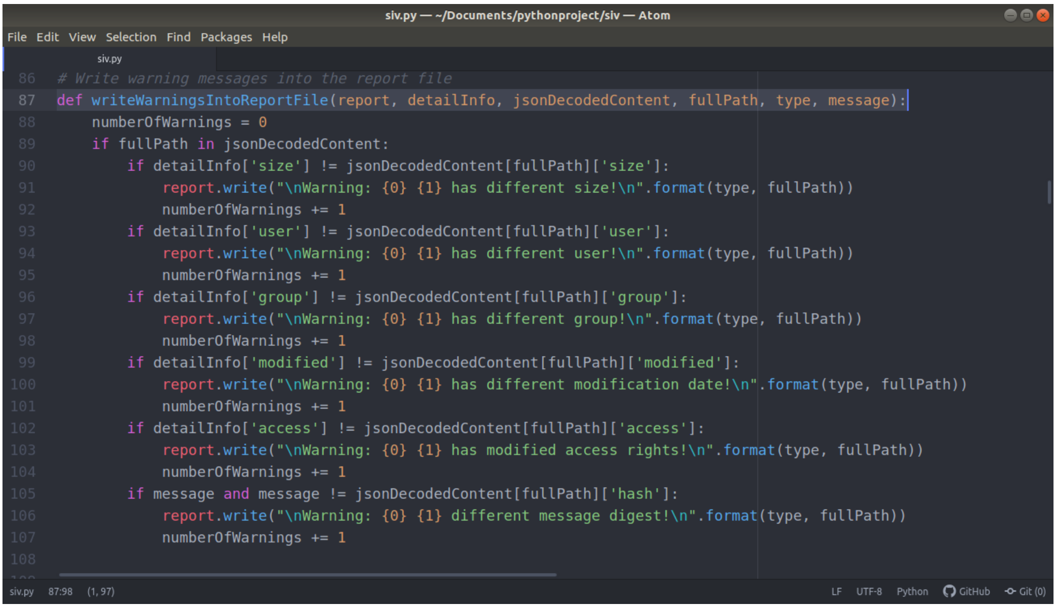
Finally write the *JSON formatted* (directory, file and hashing function details) data into the ***verification.json*** file and the report related queries into the ***report.txt*** file using the method ***writeIntoVerificationAndReportFile()***.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 155:154 (1, 153)
LF UTF-8 Python GitHub Git (0)

```
# Write into the verification and report file
def writeIntoVerificationAndReportFile(parsedArguments, numberOfParsedDirectories, numberOfParsedFiles, startTime
    if parsedArguments.initialization:
        openMode = "w";
        headerText = "Initialization Mode Started"
        footerText = "Initialization Mode Ended"

        # Write into the verification file
        print("\nVerification file has been created at the location: {0}\n".format(os.path.abspath(parsedArgument
        with open(parsedArguments.verification_file, "w") as verification:
            verification.write(jsonEncodedContent)
    else:
        openMode = "a";
        headerText = "Verification Mode Started"
        footerText = "Verification Mode Ended"

    # Write report details
    endTime = datetime.utcnow()
    with open(parsedArguments.report_file, openMode) as report:
        report.write("\n############################## {0} ##############################\n\n".format(headerText)
        report.write("Monitored directory         : {0}\n".format(os.path.abspath(parsedArguments.monitored_direc
        report.write("Verification file           : {0}\n".format(os.path.abspath(parsedArguments.verification_fi
        if parsedArguments.verification:
```

Figure: Method that writes JSON data into verification file and writes report.

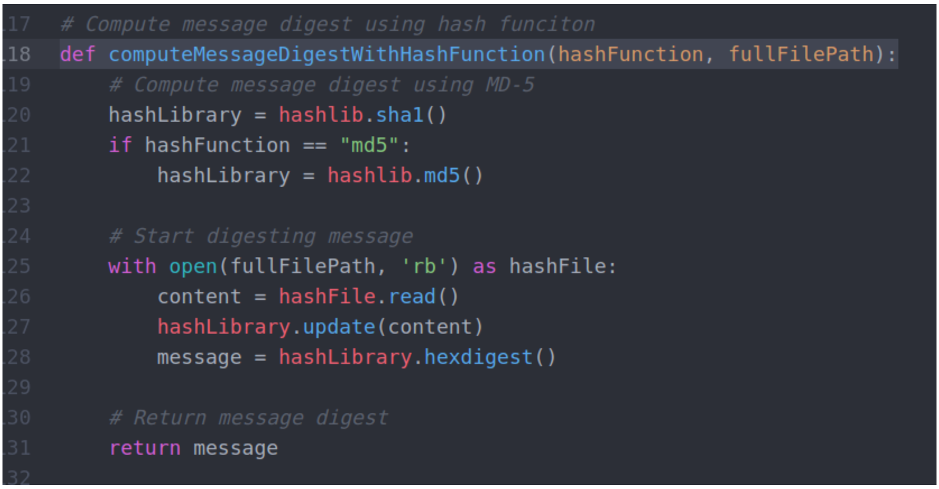
I developed a separate method, ***writeWarningsIntoReportFile()***, for writing warnings of any changes in the filesystem into the ***report.txt*** file.


siv.py — ~/Documents/pythonproject/siv — Atom
File Edit View Selection Find Packages Help
siv.py
siv.py 87:98 (1, 97)
LF UTF-8 Python GitHub Git (0)

```python
# Write warning messages into the report file
def writeWarningsIntoReportFile(report, detailInfo, jsonDecodedContent, fullPath, type, message):
    numberOfWarnings = 0
    if fullPath in jsonDecodedContent:
        if detailInfo['size'] != jsonDecodedContent[fullPath]['size']:
            report.write("\nWarning: {0} {1} has different size!\n".format(type, fullPath))
            numberOfWarnings += 1
        if detailInfo['user'] != jsonDecodedContent[fullPath]['user']:
            report.write("\nWarning: {0} {1} has different user!\n".format(type, fullPath))
            numberOfWarnings += 1
        if detailInfo['group'] != jsonDecodedContent[fullPath]['group']:
            report.write("\nWarning: {0} {1} has different group!\n".format(type, fullPath))
            numberOfWarnings += 1
        if detailInfo['modified'] != jsonDecodedContent[fullPath]['modified']:
            report.write("\nWarning: {0} {1} has different modification date!\n".format(type, fullPath))
            numberOfWarnings += 1
        if detailInfo['access'] != jsonDecodedContent[fullPath]['access']:
            report.write("\nWarning: {0} {1} has modified access rights!\n".format(type, fullPath))
            numberOfWarnings += 1
        if message and message != jsonDecodedContent[fullPath]['hash']:
            report.write("\nWarning: {0} {1} different message digest!\n".format(type, fullPath))
            numberOfWarnings += 1
```

Figure: Method that writes warnings into report file.

For computing the hashing message digest, I developed a comprehensive function, ***computeMessageDigestWithHashFunction()***, which will be reused time to time for both *Initialization* and *Verification* modes with separate arguments.



```python
# Compute message digest using hash funciton
def computeMessageDigestWithHashFunction(hashFunction, fullFilePath):
    # Compute message digest using MD-5
    hashLibrary = hashlib.sha1()
    if hashFunction == "md5":
        hashLibrary = hashlib.md5()

    # Start digesting message
    with open(fullFilePath, 'rb') as hashFile:
        content = hashFile.read()
        hashLibrary.update(content)
        message = hashLibrary.hexdigest()

    # Return message digest
    return message
```

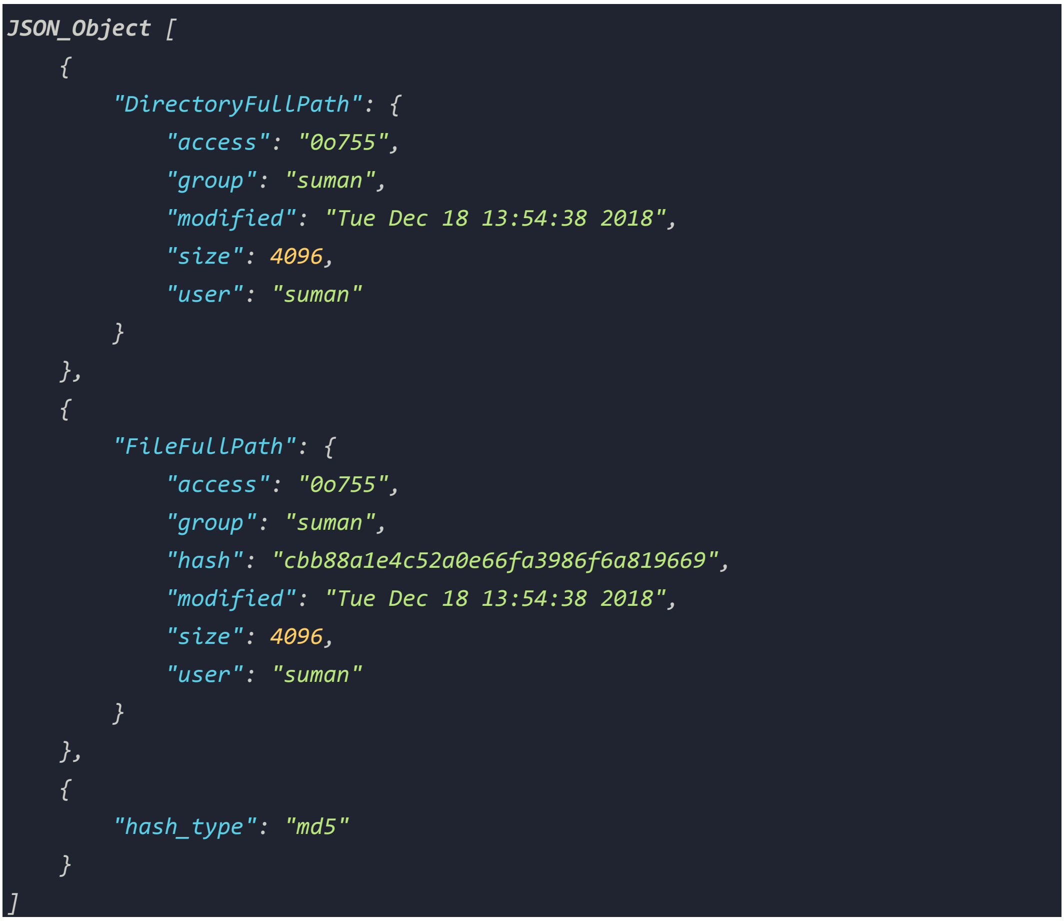
Figure: Method that computes hashing message digest.

2.3 Verification File Format

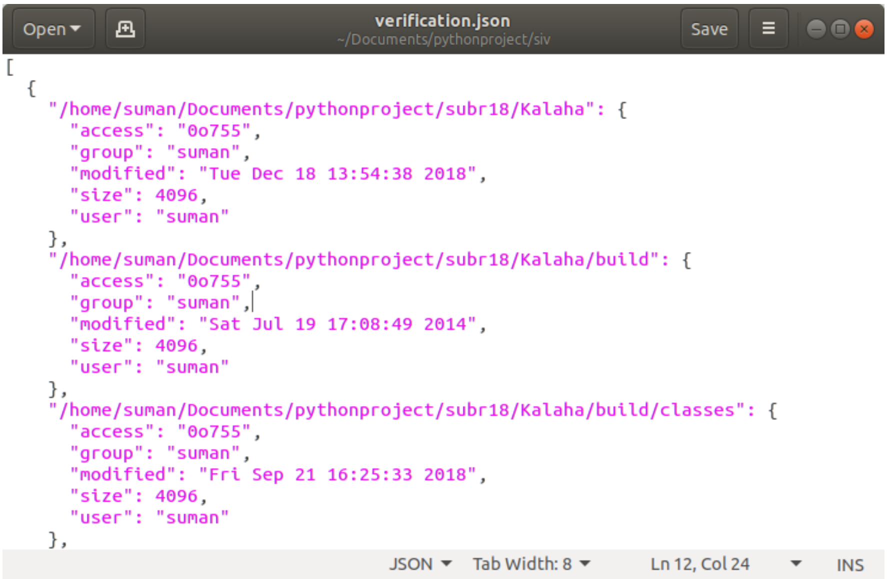
In the verification file, I'm storing all the data as *JSON (JavaScript Object Notation)* format. Because *JSON* is a one of the most feasible and convenient data formats, which is very easily readable and clearly understandable by human. Alongside, it is easy to handle and manipulate *JSON data* compare to other data format. In my verification file, I stored data in three segments.

In first segment, I stored all the details of the directories and subdirectories.
In second segment, I stored all the details of the files.
In third segment, I stored hash function information.

The key of the directory and file information is the full-path of directory and file itself as mentioned in the structure. *JSON* file structure is given below -



```
JSON_Object [
    {
        "DirectoryFullPath": {
            "access": "0o755",
            "group": "suman",
            "modified": "Tue Dec 18 13:54:38 2018",
            "size": 4096,
            "user": "suman"
        }
    },
    {
        "FileFullPath": {
            "access": "0o755",
            "group": "suman",
            "hash": "cbb88a1e4c52a0e66fa3986f6a819669",
            "modified": "Tue Dec 18 13:54:38 2018",
            "size": 4096,
            "user": "suman"
        }
    },
    {
        "hash_type": "md5"
    }
]
```



```
[
  {
    "/home/suman/Documents/pythonproject/subr18/Kalaha": {
      "access": "0o755",
      "group": "suman",
      "modified": "Tue Dec 18 13:54:38 2018",
      "size": 4096,
      "user": "suman"
    },
    "/home/suman/Documents/pythonproject/subr18/Kalaha/build": {
      "access": "0o755",
      "group": "suman",
      "modified": "Sat Jul 19 17:08:49 2014",
      "size": 4096,
      "user": "suman"
    },
    "/home/suman/Documents/pythonproject/subr18/Kalaha/build/classes": {
      "access": "0o755",
      "group": "suman",
      "modified": "Fri Sep 21 16:25:33 2018",
      "size": 4096,
      "user": "suman"
    },
```

Figure: Verification file format.

2.4 Verification File Datatype

In the verification file, I'm storing directory and file information of different *datatypes*. Below I showed the different *datatypes* I used for storing information in the verification file -

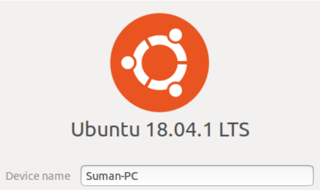
*"access": "0o755", -**Datatype: Octal-String***
*"group": "suman", -**Datatype: String***
*"hash": "cbb88a1e4c52a0e66fa3986f6a819669", - **Datatype: Hexa-String***
*"modified": "Tue Dec 18 13:54:38 2018", - **Datatype: Datetime-String***
*"size": 4096, - **Datatype: Integer***
*"user": "suman" - **Datatype: String***

2.5 Programming Language

I used programming language – *Python* for developing *System Integrity Verifier (SIV)*. *Python* is a general-purpose language and can be used to build just about anything. It is primarily used in developing solutions to complex issues within a short-time and less lines of code than many other languages. Professionally, *Python* is great for data analysis, artificial intelligence and scientific computing. Its simplicity, user-friendly features, intuitive coding style and easy use in *Data Science* convinced me to use *Python* for developing this *SIV-Application*.

2.6 Software Dependencies

I executed the *SIV Application* on ***Ubuntu 18.04.1 LTS***

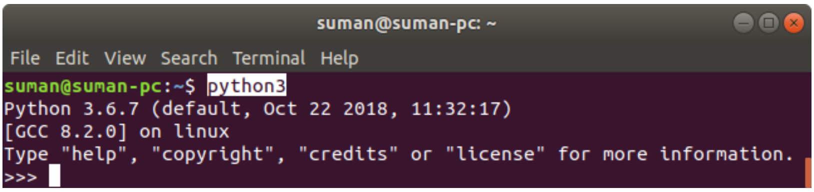


Ubuntu 18.04.1 LTS

Device name: Suman-PC

Figure: Ubuntu Version of Host

Usually *Python* comes by default with the *Ubuntu* installation. With ***Ubuntu 18.04.1 LTS*** pack, we get ***Python 2.7.15rc1***, but I developed this *SIV Application* in *Python3* environment and tested successfully in ***Python 3.6.7***. So I would recommend you to have one of these two *Python Versions* (***Python 3.6.6** or **Python 3.6.7***) on your testing system.



```
suman@suman-pc:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Figure: Python Version of Host PC

3 SIV USAGE

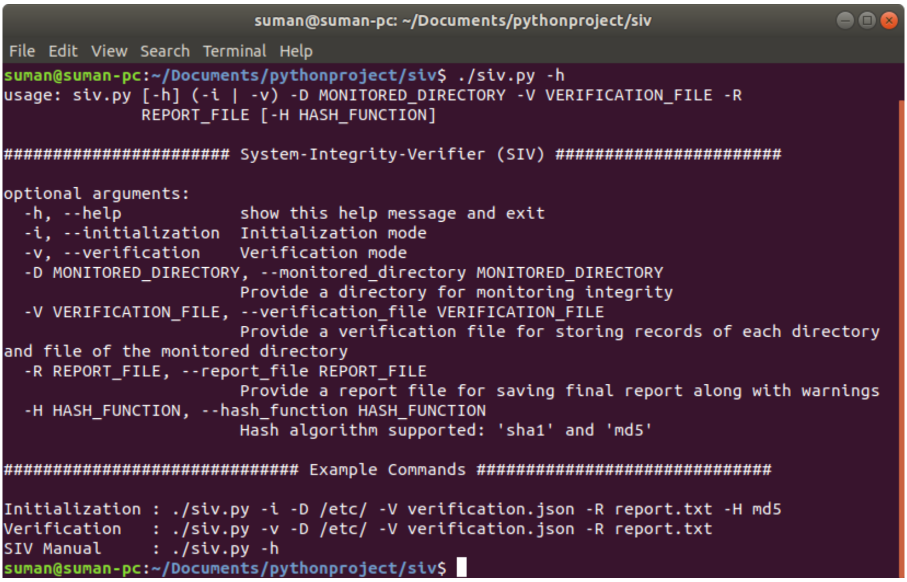
siv.py [-h] (-i | -v) -D MONITORED_DIRECTORY -V VERIFICATION_FILE -R REPORT_FILE [-H HASH_FUNCTION]

<u>Optional Arguments:</u>

-h, --help show this help message and exit
-i, --initialization Initialization mode
-v, --verification Verification mode
-D MONITORED_DIRECTORY, --monitored_directory MONITORED_DIRECTORY
Provide a directory for monitoring integrity
-V VERIFICATION_FILE, --verification_file VERIFICATION_FILE
Provide a verification file for storing records of each directory and file of the monitored directory
-R REPORT_FILE, --report_file REPORT_FILE
Provide a report file for saving final report along with warnings
-H HASH_FUNCTION, --hash_function HASH_FUNCTION
Hash algorithm supported: 'sha1' and 'md5'

<u>Sample Commands:</u>

Initialization	: *./siv.py -i -D /etc/ -V verification.json -R report.txt -H md5*
Verification	: *./siv.py -v -D /etc/ -V verification.json -R report.txt*
SIV Help Manual	: *./siv.py -h*


suman@suman-pc: ~/Documents/pythonproject/siv
File Edit View Search Terminal Help

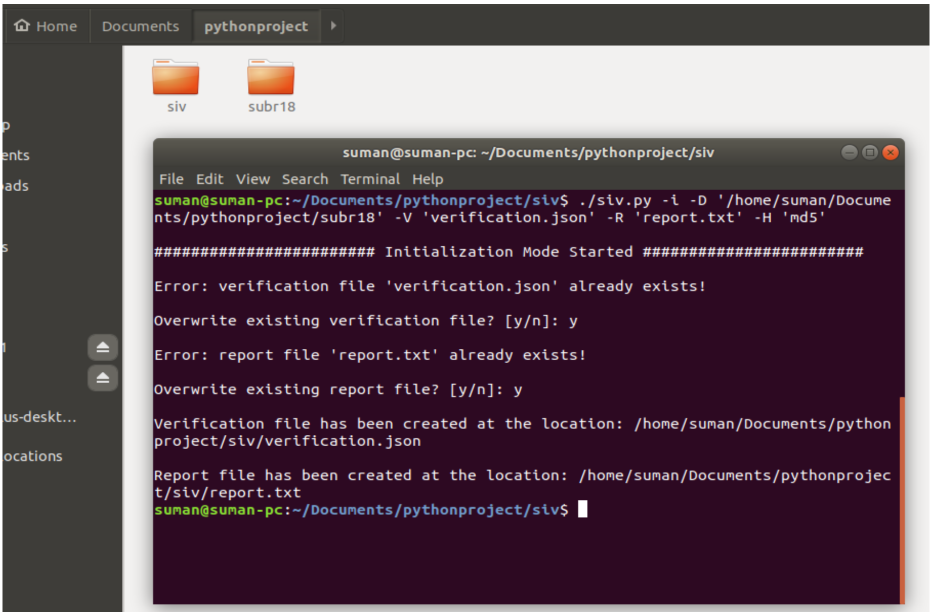
```
suman@suman-pc:~/Documents/pythonproject/siv$ ./siv.py -h
usage: siv.py [-h] (-i | -v) -D MONITORED_DIRECTORY -V VERIFICATION_FILE -R
              REPORT_FILE [-H HASH_FUNCTION]

####################### System-Integrity-Verifier (SIV) #######################

optional arguments:
  -h, --help            show this help message and exit
  -i, --initialization  Initialization mode
  -v, --verification    Verification mode
  -D MONITORED_DIRECTORY, --monitored_directory MONITORED_DIRECTORY
                        Provide a directory for monitoring integrity
  -V VERIFICATION_FILE, --verification_file VERIFICATION_FILE
                        Provide a verification file for storing records of each directory
and file of the monitored directory
  -R REPORT_FILE, --report_file REPORT_FILE
                        Provide a report file for saving final report along with warnings
  -H HASH_FUNCTION, --hash_function HASH_FUNCTION
                        Hash algorithm supported: 'sha1' and 'md5'

############################## Example Commands ##############################

Initialization : ./siv.py -i -D /etc/ -V verification.json -R report.txt -H md5
Verification   : ./siv.py -v -D /etc/ -V verification.json -R report.txt
SIV Manual     : ./siv.py -h
suman@suman-pc:~/Documents/pythonproject/siv$
```

Figure: System Integrity Verifier (SIV) Usages.

<u>Most Common Operations:</u>

I executed the *SIV application* in *Initialization* mode using following command -

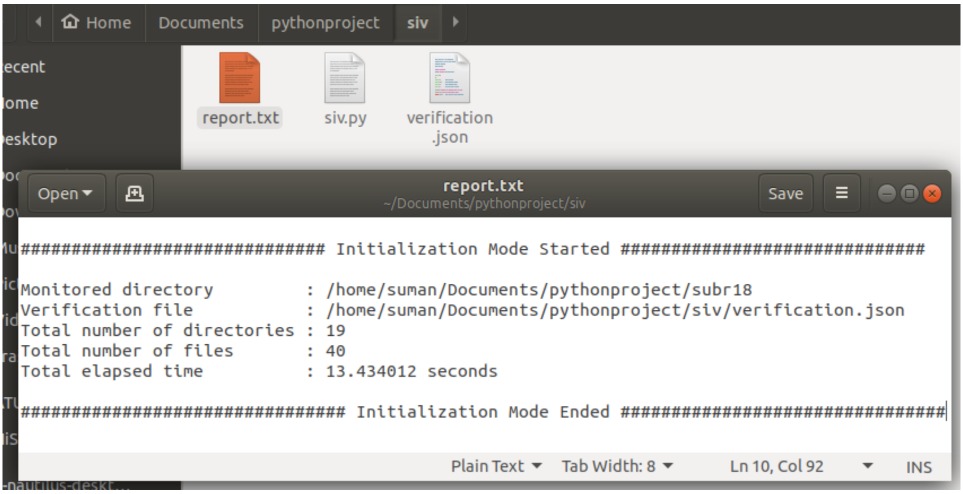
./siv.py -i -D '/home/suman/Documents/pythonproject/subr18' -V 'verification.json' -R 'report.txt' -H 'md5'


Home
Documents
pythonproject
siv
subr18
suman@suman-pc: ~/Documents/pythonproject/siv
File Edit View Search Terminal Help

```
suman@suman-pc:~/Documents/pythonproject/siv$ ./siv.py -i -D '/home/suman/Documents/pythonproject/subr18' -V 'verification.json' -R 'report.txt' -H 'md5'

######################## Initialization Mode Started ########################

Error: verification file 'verification.json' already exists!

Overwrite existing verification file? [y/n]: y

Error: report file 'report.txt' already exists!

Overwrite existing report file? [y/n]: y

Verification file has been created at the location: /home/suman/Documents/python
project/siv/verification.json

Report file has been created at the location: /home/suman/Documents/pythonprojec
t/siv/report.txt
suman@suman-pc:~/Documents/pythonproject/siv$
```

Figure: Executing SIV in Initialization Mode.

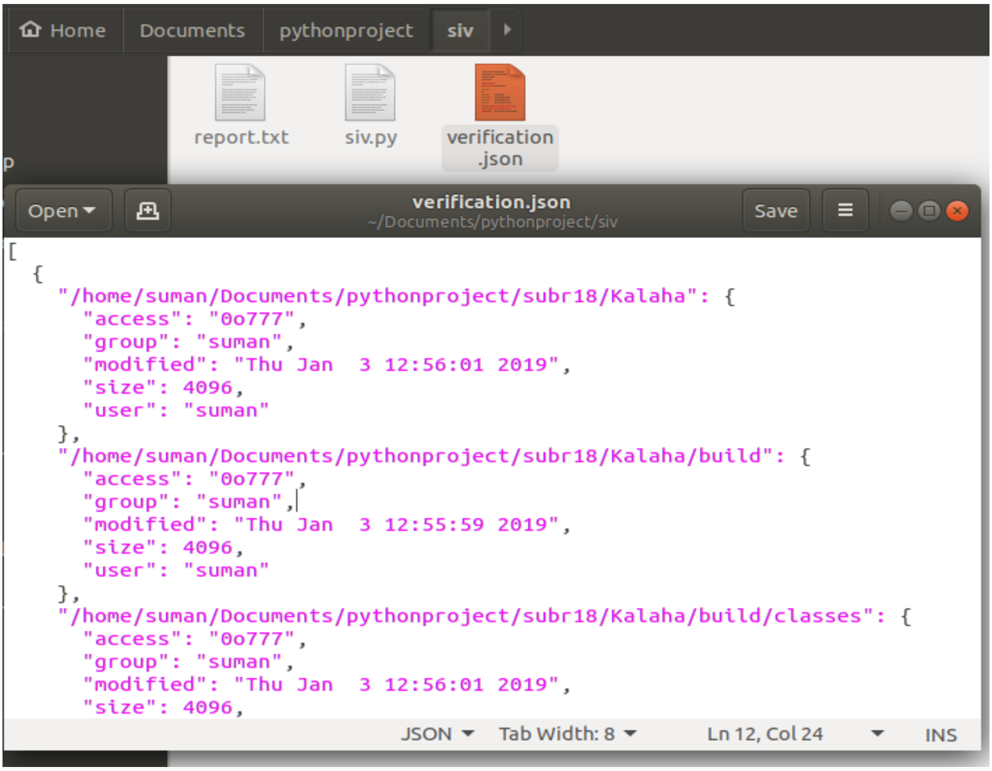
After executing the SIV application in Initialization Mode, I got Initialization Report as follows –



```
############################## Initialization Mode Started ##############################

Monitored directory         : /home/suman/Documents/pythonproject/subr18
Verification file           : /home/suman/Documents/pythonproject/siv/verification.json
Total number of directories : 19
Total number of files       : 40
Total elapsed time          : 13.434012 seconds

################################ Initialization Mode Ended ################################
```

Figure: SIV Initialization Report.

After executing the *SIV application* in *Initialization Mode*, I got following directories, subdirectories and files information into the verification file -

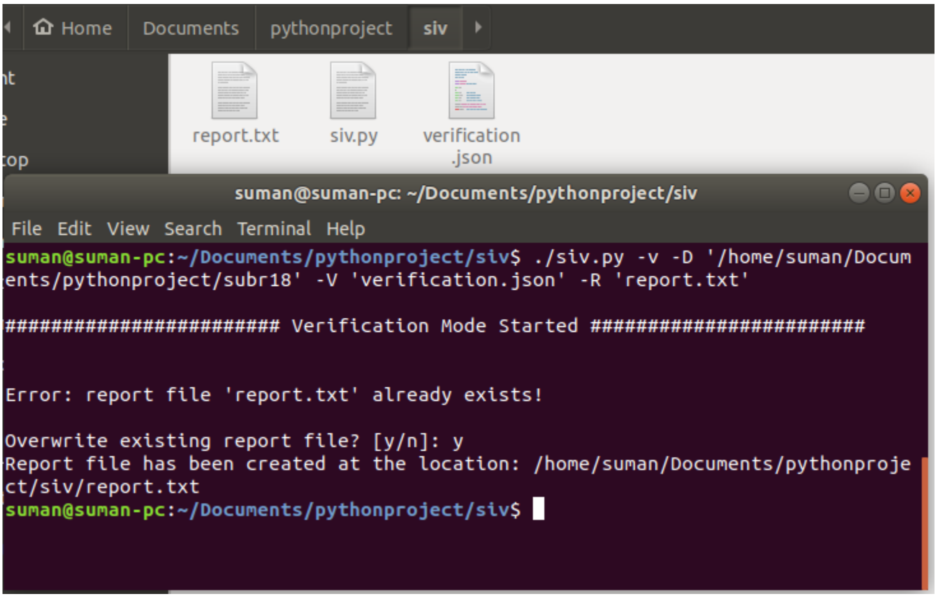


```
[
  {
    "/home/suman/Documents/pythonproject/subr18/Kalaha": {
      "access": "0o777",
      "group": "suman",
      "modified": "Thu Jan  3 12:56:01 2019",
      "size": 4096,
      "user": "suman"
    },
    "/home/suman/Documents/pythonproject/subr18/Kalaha/build": {
      "access": "0o777",
      "group": "suman",
      "modified": "Thu Jan  3 12:55:59 2019",
      "size": 4096,
      "user": "suman"
    },
    "/home/suman/Documents/pythonproject/subr18/Kalaha/build/classes": {
      "access": "0o777",
      "group": "suman",
      "modified": "Thu Jan  3 12:56:01 2019",
      "size": 4096,
```

Figure: Verification data after SIV Initialization.

Then I executed the *SIV application* in *Verification* mode using following command -

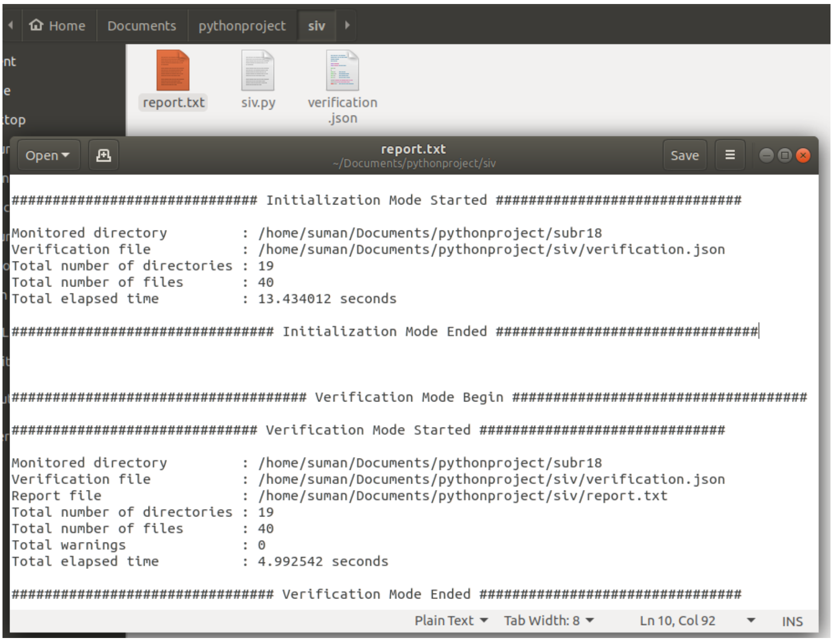
./siv.py -v -D '/home/suman/Documents/pythonproject/subr18' -V 'verification.json' -R 'report.txt'



```
suman@suman-pc:~/Documents/pythonproject/siv$ ./siv.py -v -D '/home/suman/Docum
ents/pythonproject/subr18' -V 'verification.json' -R 'report.txt'

######################## Verification Mode Started ########################


Error: report file 'report.txt' already exists!

Overwrite existing report file? [y/n]: y
Report file has been created at the location: /home/suman/Documents/pythonproje
ct/siv/report.txt
suman@suman-pc:~/Documents/pythonproject/siv$ 
```

Figure: Executing SIV in Verification Mode.

After executing the *SIV application* in *Verification Mode*, I got *Verification Report* without warning message as follows –

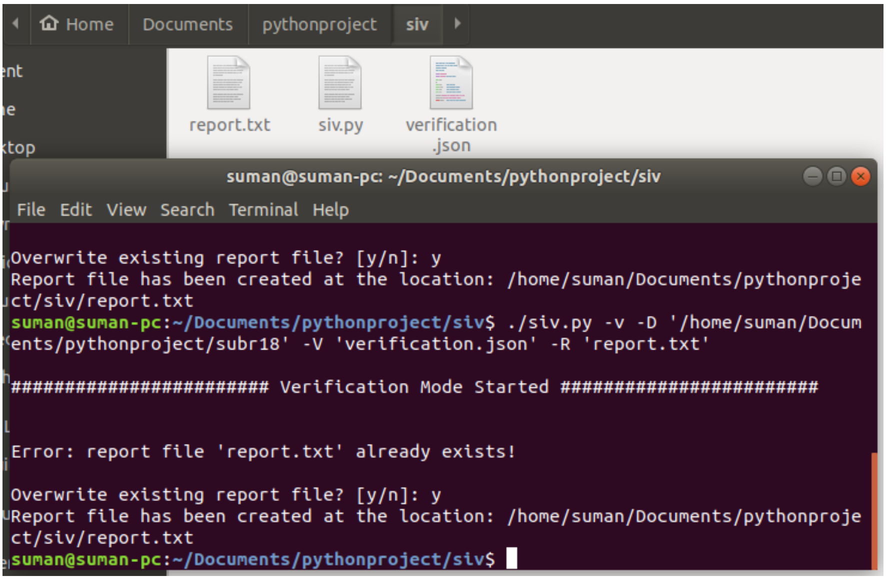


```
############################## Initialization Mode Started ##############################

Monitored directory          : /home/suman/Documents/pythonproject/subr18
Verification file            : /home/suman/Documents/pythonproject/siv/verification.json
Total number of directories : 19
Total number of files        : 40
Total elapsed time           : 13.434012 seconds

################################ Initialization Mode Ended ################################



#################################### Verification Mode Begin ####################################

############################## Verification Mode Started ##############################

Monitored directory          : /home/suman/Documents/pythonproject/subr18
Verification file            : /home/suman/Documents/pythonproject/siv/verification.json
Report file                  : /home/suman/Documents/pythonproject/siv/report.txt
Total number of directories : 19
Total number of files        : 40
Total warnings               : 0
Total elapsed time           : 4.992542 seconds

################################ Verification Mode Ended ################################
```

Figure: SIV Verification Report (without warning).

Then I changed inside monitored directory and executed the *SIV application* in *Verification* mode again using following command to see how are the changes warning messages coming along -

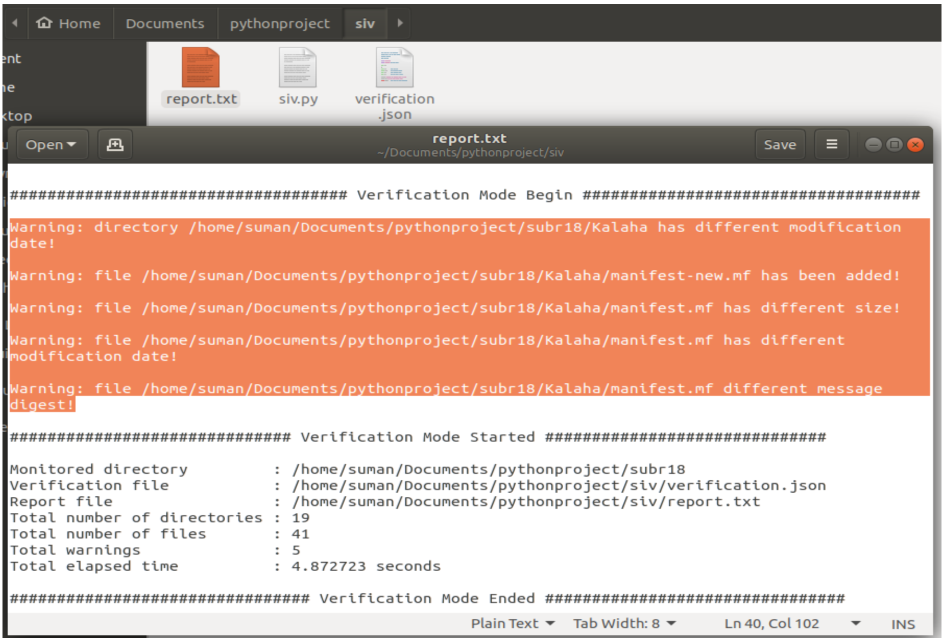
./siv.py -v -D '/home/suman/Documents/pythonproject/subr18' -V 'verification.json' -R 'report.txt'



```
Overwrite existing report file? [y/n]: y
Report file has been created at the location: /home/suman/Documents/pythonproje
ct/siv/report.txt
suman@suman-pc:~/Documents/pythonproject/siv$ ./siv.py -v -D '/home/suman/Docum
ents/pythonproject/subr18' -V 'verification.json' -R 'report.txt'

######################## Verification Mode Started ########################


Error: report file 'report.txt' already exists!

Overwrite existing report file? [y/n]: y
Report file has been created at the location: /home/suman/Documents/pythonproje
ct/siv/report.txt
suman@suman-pc:~/Documents/pythonproject/siv$ 
```

Figure: Executing SIV in Verification Mode (After Modifying Filessytem).

After executing the *SIV application* again in *Verification Mode*, I got *Verification Report* with warnings message as follows –

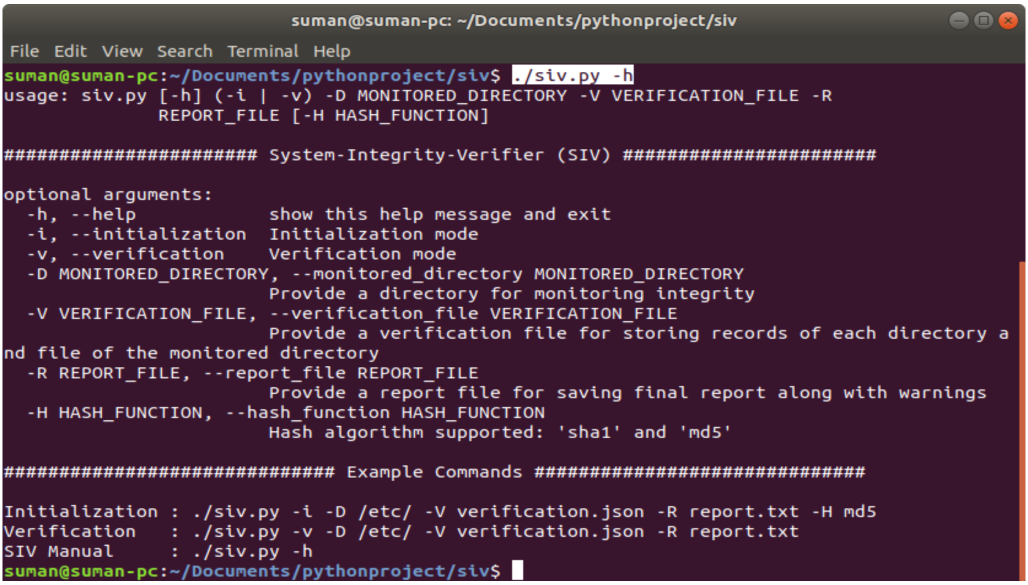


```
#################################### Verification Mode Begin ####################################

Warning: directory /home/suman/Documents/pythonproject/subr18/Kalaha has different modification
date!

Warning: file /home/suman/Documents/pythonproject/subr18/Kalaha/manifest-new.mf has been added!

Warning: file /home/suman/Documents/pythonproject/subr18/Kalaha/manifest.mf has different size!

Warning: file /home/suman/Documents/pythonproject/subr18/Kalaha/manifest.mf has different
modification date!

Warning: file /home/suman/Documents/pythonproject/subr18/Kalaha/manifest.mf different message
digest!

############################## Verification Mode Started ##############################

Monitored directory         : /home/suman/Documents/pythonproject/subr18
Verification file           : /home/suman/Documents/pythonproject/siv/verification.json
Report file                 : /home/suman/Documents/pythonproject/siv/report.txt
Total number of directories : 19
Total number of files       : 41
Total warnings              : 5
Total elapsed time          : 4.872723 seconds

################################ Verification Mode Ended ################################
```

Figure: SIV Verification Report (with warnings).

I executed following command on terminal for *SIV* helper manual -

./siv.py -h


suman@suman-pc: ~/Documents/pythonproject/siv
File Edit View Search Terminal Help

```
suman@suman-pc:~/Documents/pythonproject/siv$ ./siv.py -h
usage: siv.py [-h] (-i | -v) -D MONITORED_DIRECTORY -V VERIFICATION_FILE -R
              REPORT_FILE [-H HASH_FUNCTION]

####################### System-Integrity-Verifier (SIV) #######################

optional arguments:
  -h, --help            show this help message and exit
  -i, --initialization  Initialization mode
  -v, --verification    Verification mode
  -D MONITORED_DIRECTORY, --monitored_directory MONITORED_DIRECTORY
                        Provide a directory for monitoring integrity
  -V VERIFICATION_FILE, --verification_file VERIFICATION_FILE
                        Provide a verification file for storing records of each directory a
nd file of the monitored directory
  -R REPORT_FILE, --report_file REPORT_FILE
                        Provide a report file for saving final report along with warnings
  -H HASH_FUNCTION, --hash_function HASH_FUNCTION
                        Hash algorithm supported: 'sha1' and 'md5'

############################## Example Commands ##############################

Initialization : ./siv.py -i -D /etc/ -V verification.json -R report.txt -H md5
Verification   : ./siv.py -v -D /etc/ -V verification.json -R report.txt
SIV Manual     : ./siv.py -h
suman@suman-pc:~/Documents/pythonproject/siv$ 
```

Figure: SIV Helper Manual.