

Assignment 2: System Integrity Verifier(SIV)
Network and System Security - ET2595
Master program in computer science
17-01-2024

ET2595:Network and System Security

Assignment 2: System Integrity Verifier(SIV)

Monica Gattupalli

Personal number: 199911305002

E-mail: moga20@student.bth.se

List of Figures

1.1	Libraries that needed for SIV	2
1.2	file verifier	2
1.3	file utilities	3
1.4	Process directory data	3
1.5	Traverse directory	4
1.6	Report generator	4
1.7	Genearte verification	5
1.8	Report verification	5
1.9	Arguments of SIV	6
2.1	Output of SIV	7
2.2	output of SIV	7
2.3	Output of SIV	8
2.4	output of initialization mode	8
2.5	output of initialization mode	9
2.6	output of verification mode	9
2.7	Modification of file	10
2.8	Output of verification with last modification	10

Contents

List of Figures	i
1 Introduction	1
1.1 Initialization mode	1
1.2 Verification mode	1
1.3 Design and implementation	1
1.4 Usage	6
2 Result	7

Chapter 1

Introduction

The goal of this assignment is to implement an SIV for a Linux system that detects modifications that occur in a file system within a directory tree. In SIV there are two modes Initialization mode and verification mode and a detailed explanation of the are given below.

1.1 Initialization mode

In this mode, according to the user path, the directory is retrieved, and the hash function is computed based on the selection of the user during this it collects the information related to the file like the owner, size of the file, and access rights, and stores the information in JSON file. A report is generated which contains information about the parsed files, and directories. the outputs of this mode are presented in the results section.

1.2 Verification mode

This mode makes a comparison of the files with the current file identifies the modifications made to the updated file and also collects information like the size of the file, last modified time, owner, and hash value. with all these, a report is generated and stored in JSON format. The output of this mode is presented in the results section.

1.3 Design and implementation

To implement a system integrity verifier (SIV), the python programming language is used. this programming language is user-friendly and has many libraries that support users in every accept.

In this section designed code for SIV is discussed in detail.

In the below code, all the libraries that are needed to run the SIV are imported.

```

import os
import argparse
import sys
import hashlib
import argparse
import sys
import grp
import pwd
import time
import json
from pathlib import Path
from datetime import datetime

```

Figure 1.1: Libraries that needed for SIV

The below class is a File verifier that checks the file location within target directory. This class has an instance method "does_directory_exist" that checks the directory files based on the path, and a "check_file_directory" method to check the file path. and another method "verify_files_location" is used to verify the paths and return True if they are located otherwise returns False.

```

class FileVerifier:
    def __init__(self, target_dir):
        self.target_dir = target_dir

    def does_directory_exist(dir_name):
        # Using os.path.exists to check directory existence
        return os.path.exists(dir_name)

    def check_file_directory(self, file_path):
        file_base_name = os.path.basename(file_path)

        if self.target_dir in file_base_name:
            print(f"ERROR: The file '{os.path.basename(file_path)}' is within the target directory")
            return False

        return True

    @staticmethod
    def verify_files_location(target_dir, verification_file_path, report_file_path):
        verifier = FileVerifier(target_dir)

        if not verifier.check_file_directory(verification_file_path):
            return False # Verification file is inside the target directory

        if not verifier.check_file_directory(report_file_path):
            return False # Report file is inside the target directory

        return True

```

Figure 1.2: file verifier

In the below-attached code a class called "Fileutilities" helps in handling the file operation and supports two different hash functions "sha1" and "md5". there is a method in the class to verify whether the user entered hash function supported or not, there is another method that overwrites the file (by either creating or truncating the file based on its existence).

```

class FileUtilities:
    LIST_OF_SUPPORTED_HASH_FUNCTIONS = ["sha1", "md5"]

    def __init__(self):
        pass # Constructor, you can add initialization logic here if need

    def is_requested_hash_function_okay(self, requested_hash_technique):
        return requested_hash_technique in self.LIST_OF_SUPPORTED_HASH_FUNCTIONS

    def overwrite_file(self, file_to_overwrite):
        try:
            with open(file_to_overwrite, "w"):
                pass # This will create or truncate the file if it exists
        except Exception as exception_message:
            print(f"Error: {exception_message}")

```

Figure 1.3: file utilities

In the below-attached part of code is called "Process_directory_data" is used to gather and calculate many details about the files based on the absolute file path and hash function. The OS library is used to extract statistical and ownership information of the file like size, owner, access rights, last modified, and hash function. "calculate_file_hash" is the method used to calculate the hash value based on the hash function selected.

```

def process_directory_data(absolute_file_path_to_process, selected_hashing_function):
    file_statistics_information = os.stat(absolute_file_path_to_process)
    user_identifier = file_statistics_information.st_uid
    group_identifier = file_statistics_information.st_gid
    username_of_owner = pwd.getpwuid(user_identifier)[0] # extracting the username
    groupname_of_owner = grp.getgrgid(group_identifier)[0] # extracting the groupname

    def calculate_file_hash(path_to_file_or_directory, chosen_hashing_algorithm):
        hash_object = hashlib.shai() if chosen_hashing_algorithm == "sha1" else hashlib.md5()
        if os.path.isdir(path_to_file_or_directory):
            hash_object.update((path_to_file_or_directory * 1024).encode("utf-8"))
        else:
            with open(path_to_file_or_directory, 'rb') as file_to_hash:
                while True:
                    block = file_to_hash.read(2**10)
                    if not block:
                        break
                    hash_object.update(block) if hash_object else None
            return hash_object.hexdigest() if hash_object else ""

    file_information_details = {
        "FILE_SIZE": str(file_statistics_information.st_size),
        "FILE_USER_OWNER": username_of_owner,
        "FILE_GROUP_OWNER": groupname_of_owner,
        "FILE_ACCESS_RIGHTS": str(file_statistics_information.st_mode),
        "FILE_LAST_MODIFIED": str(file_statistics_information.st_mtime),
        "FILE_HASH": calculate_file_hash(absolute_file_path_to_process, selected_hashing_function)
    }

    current_file_information_storage[absolute_file_path_to_process] = file_information_details

```

Figure 1.4: Process directory data

In the below-attached part of the code is "traverse_directory_get" It takes the name of the directory to search and, the mode of operation requested as input. it keeps a count of the files and directories it processed. if the selected mode of operation is valid then the file or director processes the data according to and updates the count. for each traversal, the global variable is incremented.

```

def traverse_directory_and_get(directory_name_to_search, requested_operation_mode):
    global total_number_of_files_parsed, total_number_of_directories_parsed

    # Dictionary to store counts of files and directories
    counts = {'files': 0, 'directories': 0}

    # Determine if the requested operation mode is supported
    supported_operation_mode = requested_operation_mode if requested_operation_mode in ['f', 'd'] else None

    process_directory_data(directory_name_to_search, chosen_hashing_algorithm)

    # Fetch sorted file entries in the directory using pathlib.Path
    directory_path = Path(directory_name_to_search)
    entries_to_process = sorted(directory_path.iterdir(), key=lambda x: x.name)

    for current_entry in entries_to_process:
        current_path_to_process = str(current_entry) # Convert Path object to string
        if current_entry.is_dir():
            traverse_directory_and_get(current_path_to_process, requested_operation_mode)
            counts['directories'] += 1
        elif current_entry.is_file():
            counts['files'] += 1 if supported_operation_mode else 0
            process_directory_data(current_path_to_process, chosen_hashing_algorithm)
        else:
            assert False, f"Skipping {current_path_to_process}: Entry is not a file or directory"

    # Update the global counts after traversal
    total_number_of_files_parsed += counts['files']
    total_number_of_directories_parsed += counts['directories']

```

Figure 1.5: Traverse directory

In the attached part of the code is the class "Report generator" which is mainly used for generating reports of the files or directories related to their monitoring and verification process. This class has three where one is used to initialization report which takes care of the details of no. of directories parsed, files parsed and time taken. the second method which is a verification report along with the directories, and files parsed, will take care of no of warnings, and time taken for verification. in the third method, it append the content and generate the report.

```

class ReportGenerator:
    def __init__(self, absolute_path_of_monitored_directory, absolute_path_of_verification_file, absolute_path_of_report_file):
        self.absolute_path_of_monitored_directory = absolute_path_of_monitored_directory
        self.absolute_path_of_verification_file = absolute_path_of_verification_file
        self.absolute_path_of_report_file = absolute_path_of_report_file

    def generate_initialization_report(self, total_number_of_directories_parsed, total_number_of_files_parsed, time_taken_for_initialization):
        report_content = (
            f"Observed Location Path : {os.path.abspath(self.absolute_path_of_monitored_directory)}\n"
            f"Verification File Path : {os.path.abspath(self.absolute_path_of_verification_file)}\n"
            f"Count of Examined Directories : {total_number_of_directories_parsed}\n"
            f"Count of Examined Files : {total_number_of_files_parsed}\n"
            f"Initiation Duration : {time_taken_for_initialization} seconds\n"
        )
        self._write_report_content(report_content)

    def generate_verification_report(self, total_number_of_directories_parsed, total_number_of_files_parsed, time_taken_for_verification, total_number_of_warnings):
        report_content = (
            f"Observed Location Path : {os.path.abspath(self.absolute_path_of_monitored_directory)}\n"
            f"Verification File Path : {os.path.abspath(self.absolute_path_of_verification_file)}\n"
            f"Outcome File Path : {os.path.abspath(self.absolute_path_of_report_file)}\n"
            f"Count of Examined Directories : {total_number_of_directories_parsed}\n"
            f"Count of Examined Files : {total_number_of_files_parsed}\n"
            f"Count of Warnings Issued : {total_number_of_warnings}\n"
            f"Verification Duration : {time_taken_for_verification} seconds\n"
        )
        self._write_report_content(report_content)

    def _write_report_content(self, content_to_write):
        with open(os.path.abspath(self.absolute_path_of_report_file), "a") as file_pointer:
            file_pointer.write(content_to_write)
        print(content_to_write)

```

Figure 1.6: Report generator

In the attached part of the code is the method "generate_verification_report" which compares the information in the current file and the verification document to identify the modifications. converts the time stamps to human-readable format.

In the attached part of the code is the method "report_verification_mode"

```
def generate_verification_report():
    global warnings_given
    verification_details = {}

    for inspected_path_or_document, current_data in current_file_information_storage.items():
        verification_data = verification_document_data.get(inspected_path_or_document)

        if verification_data:
            modified_content = {}

            attributes_to_inspect = [
                'FILE_SIZE', 'FILE_USER_OWNER', 'FILE_GROUP_OWNER',
                'FILE_ACCESS_RIGHTS', 'FILE_LAST_MODIFIED', 'FILE_HASH'
            ]

            for attribute in attributes_to_inspect:
                current_value = current_data.get(attribute)
                verification_value = verification_data.get(attribute)

                if verification_value != current_value:
                    modified_content.update({
                        f"{NEW_ATTR_PREFIX}{attribute}": current_value,
                        f"{OLD_ATTR_PREFIX}{attribute}": verification_value
                    })
                    warnings_given += 1

            if modified_content:
                verification_details[inspected_path_or_document] = modified_content
            else:
                verification_details[inspected_path_or_document] = current_data

    for missing_entity in verification_document_data.keys():
        if missing_entity not in current_file_information_storage and missing_entity != '':
            deleted_entity_data = {}
```

Figure 1.7: Genearte verification

At first it checks if the file exists or not through `os.path`, if true then loads the JSON file and verifies the files. by using the lambda function it retrieves hash function and load data, if this is successful then the verification mode setup is complete.

```
def report_verification_mode(verification_path):
    global verification_document_data, chosen_hashing_algorithm

    # Confirm the existence of the file using the 'path.exists' method directly
    assert os.path.exists(verification_path), f"The specified verification file '{verification_path}' does not exist"

    with open(verification_path, "r") as file_handle:
        # Load JSON data from the file
        verification_document_data = json.load(file_handle)

    # Lambda function using 'get' to retrieve the hash function from the loaded data
    retrieve_hash_function = lambda information: information.get(HASH_FUNCTION)

    # Retrieve the hash function and ensure its presence in the loaded data
    chosen_hashing_algorithm = retrieve_hash_function(verification_document_data)
    assert chosen_hashing_algorithm is not None, "The verification file is invalid; hash function not found"

    return True
```

Figure 1.8: Report verification

The final art of the code and where the flow starts is the main it has parsing arguments they are

1. -i: enable the initialization mode
2. -v: enable the verification mode.

3. -D: monitory directory
4. -V: verification of file
5. -R: report -file
6. -H: hash function

```
group = parser.add_mutually_exclusive_group(required=True)
group.add_argument(
    '-i',
    action='store_true',
    dest='initialization_mode',
    help='Enable the INITIALIZATION Mode'
)
group.add_argument(
    '-v',
    action='store_true',
    dest='verification_mode',
    help='Enable the VERIFICATION mode'
)

# Define arguments separately
arguments = [
    ('-D', 'monitored_directory', str, 'Directory to be Monitored'),
    ('-V', 'verification_file', str, 'Verification Database (DB) file in .CSV format'),
    ('-R', 'report_file', str, 'Report file name'),
    ('-H', 'hash_function', str, 'Hashing Algorithm (used only for Initialization Mode), SUPPORTED_HASH_FUNCTIONS)
]
```

Figure 1.9: Arguments of SIV

1.4 Usage

As mentioned in the design and implementation section there are 6 arguments where 2 arguments are about specifying the mode that needs to be enabled and the remaining 4 are support arguments that help to monitor, report generator, and hash function.

The below are examples of the usage of the arguments to execute the SIV.

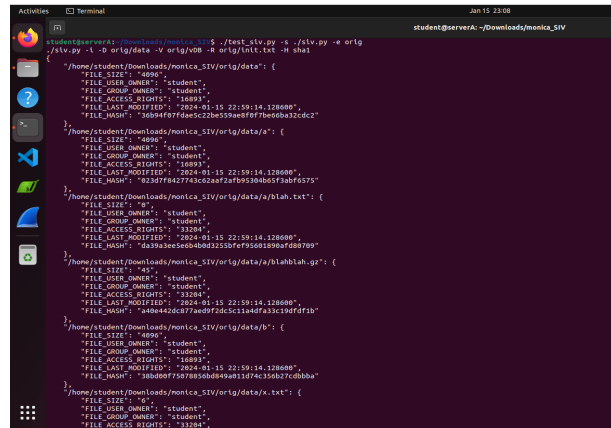
- `"./siv.py -i -D process_dir/ -V verification_db.json -R report_init.txt -H sha1"`
- `"./siv.py -v -D process_dir/ -V verification_db.json -R verification_report.txt"`.

Chapter 2

Result

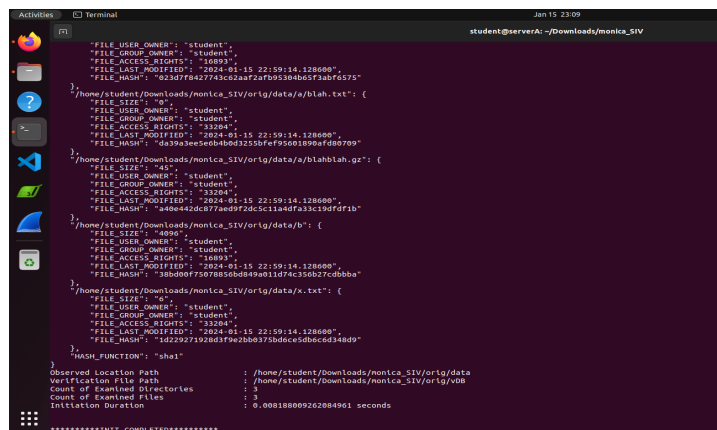
In this section, the results of SIV are discussed and the relevant pictures are attached.

The command used to test the file `./test_siv.py -s ./siv.py -e orig`.



```
student@server:~$ ./test_siv.py -s ./siv.py -e orig
{
  "/home/student/downloads/monica_SIV/orig/data": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "30094f9f7d9e9c23e359eeff7bde0a32dc2"
  },
  "/home/student/downloads/monica_SIV/orig/data/a": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "623f7f6427f40c3a72f8b9520b5f2a8f6575"
  },
  "/home/student/downloads/monica_SIV/orig/data/a/blah.txt": {
    "FILE_SIZE": "0",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "4a3b9ae5eb4b061235f9f95013804fd8709"
  },
  "/home/student/downloads/monica_SIV/orig/data/a/blahlah.gz": {
    "FILE_SIZE": "45",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "a40e442dc877aed9f2dc5c11a40fa33c9df0f1b"
  },
  "/home/student/downloads/monica_SIV/orig/data/b": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "38d00f758785b0bd49a011d74c35b27cd0bba"
  },
  "/home/student/downloads/monica_SIV/orig/data/x.txt": {
    "FILE_SIZE": "0",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204"
  }
}
```

Figure 2.1: Output of SIV



```
student@server:~$ ./test_siv.py -s ./siv.py -e orig
{
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "16893",
  "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
  "FILE_HASH": "30094f9f7d9e9c23e359eeff7bde0a32dc2"
},
  "/home/student/downloads/monica_SIV/orig/data/a/blah.txt": {
    "FILE_SIZE": "0",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "4a3b9ae5eb4b061235f9f95013804fd8709"
  },
  "/home/student/downloads/monica_SIV/orig/data/a/blahlah.gz": {
    "FILE_SIZE": "45",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "a40e442dc877aed9f2dc5c11a40fa33c9df0f1b"
  },
  "/home/student/downloads/monica_SIV/orig/data/b": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "38d00f758785b0bd49a011d74c35b27cd0bba"
  },
  "/home/student/downloads/monica_SIV/orig/data/x.txt": {
    "FILE_SIZE": "0",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "33204",
    "FILE_LAST_MODIFIED": "2024-01-15 22:59:14.128680",
    "FILE_HASH": "1d229271928d3f9e2b06375bdce5dbecd348d9"
  },
  "HASH_FUNCTION": "sha1"
},
  "Observed Location Path": "/home/student/downloads/monica_SIV/orig/data",
  "Verification File Path": "/home/student/downloads/monica_SIV/orig/odb",
  "Count of Examined Directories": 3,
  "Count of Examined Files": 3,
  "Initiation Duration": 0.008188009262084961 seconds
}
*****INIT COMPLETED*****
```

Figure 2.2: output of SIV

```

Count of Examined Files      : 3
Initiation Duration         : 0.008188009262084961 seconds

*****INIT COMPLETED*****
./siv.py -v -D orig/data -V orig/vDB -R orig/verify.txt
{
  "/home/student/Downloads/monica_SIV/orig/data": {
    "NEW_FILE_LAST_MODIFIED": "1705355954.4726193",
    "OLD_FILE_LAST_MODIFIED": "1705355954.1286004"
  },
  "/home/student/Downloads/monica_SIV/orig/data/a": {
    "NEW_FILE_ACCESS_RIGHTS": "16895",
    "OLD_FILE_ACCESS_RIGHTS": "16893",
    "NEW_FILE_LAST_MODIFIED": "1705355954.4726193",
    "OLD_FILE_LAST_MODIFIED": "1705355954.1286004"
  },
  "/home/student/Downloads/monica_SIV/orig/data/x.txt": {
    "NEW_FILE_SIZE": "4",
    "OLD_FILE_SIZE": "0",
    "NEW_FILE_USER_OWNER": "bob",
    "OLD_FILE_USER_OWNER": "student",
    "NEW_FILE_GROUP_OWNER": "alice",
    "OLD_FILE_GROUP_OWNER": "student",
    "NEW_FILE_LAST_MODIFIED": "0.0",
    "OLD_FILE_LAST_MODIFIED": "1705355954.1286004",
    "NEW_FILE_HASH": "6fcea05b38b1e4ce581d2de2dd898ef7f4d2563b",
    "OLD_FILE_HASH": "1d29271928d3f9e2bb0375bdce5dbcd348d9"
  },
  "/home/student/Downloads/monica_SIV/orig/data/a/blahlah.gz": {
    "DELETED_ENTITY": "/home/student/Downloads/monica_SIV/orig/data/a/blahlah.gz"
  },
  "/home/student/Downloads/monica_SIV/orig/data/b": {
    "DELETED_ENTITY": "/home/student/Downloads/monica_SIV/orig/data/b"
  }
}
Observed Location Path      : /home/student/Downloads/monica_SIV/orig/data
Verification File Path      : /home/student/Downloads/monica_SIV/orig/vDB
Outcome File Path           : /home/student/Downloads/monica_SIV/orig/verify.txt
Count of Examined Directories : 2
Count of Examined Files      : 2
Count of Warnings Issued     : 10
Verification Duration        : 0.0007886886596679688 seconds

*****VERIFY COMPLETED*****
COMPLETED

```

Figure 2.3: Output of SIV

Initialization mode result

The following command is used to get the output of the initialization mode
`./siv.py -i -D process_dir/ -V verification_db.json -R report_init.txt -H sha1`

```

student@server:~/Downloads/monica_SIV$ ./siv.py -i -D process_dir/ -V verification_db.json -R report_init.txt -H sha1
{
  "/home/student/Downloads/monica_SIV/process_dir": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 16:42:14",
    "FILE_HASH": "e57f4e2c646d5699638338438db7eccc8c3a37"
  },
  "/home/student/Downloads/monica_SIV/process_dir/A1.txt": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2023-12-23 19:38:22",
    "FILE_HASH": "5d6db74f2baad03d9297524deba72553a32ebbb8"
  },
  "/home/student/Downloads/monica_SIV/process_dir/data": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 16:42:14",
    "FILE_HASH": "88b9e73eb32a11a0a74e37a3b10a41ccc708a5d"
  },
  "/home/student/Downloads/monica_SIV/process_dir/data/sample": {
    "FILE_SIZE": "4096",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "16893",
    "FILE_LAST_MODIFIED": "2024-01-15 16:42:14",
    "FILE_HASH": "aee46b74ddaf9abae0f1243018833bdcda415b94"
  },
  "/home/student/Downloads/monica_SIV/process_dir/data/sample/1.txt": {
    "FILE_SIZE": "26",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "13204",
    "FILE_LAST_MODIFIED": "2023-12-21 18:34:18",
    "FILE_HASH": "8824f550e15de94013b70f2594f8bc74ee7f45a"
  },
  "/home/student/Downloads/monica_SIV/process_dir/data/text": {
    "FILE_SIZE": "36",
    "FILE_USER_OWNER": "student",
    "FILE_GROUP_OWNER": "student",
    "FILE_ACCESS_RIGHTS": "13204",
    "FILE_LAST_MODIFIED": "2024-01-15 14:51:42",
  }
}

```

Figure 2.4: output of initialization mode

```

student@serverA: ~/Downloads/monica_SIV
{
  "FILE_SIZE": "4096",
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "16893",
  "FILE_LAST_MODIFIED": "2024-01-15 16:42:14",
  "FILE_HASH": "6809e73e032a11a6a74e37a3b1a641c6e708a5d"
},
"/home/student/Downloads/monica_SIV/process_dir/data/sample": {
  "FILE_SIZE": "4096",
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "16893",
  "FILE_LAST_MODIFIED": "2024-01-15 16:42:14",
  "FILE_HASH": "aee46b74dda9abaef1243018835bdca413b94"
},
"/home/student/Downloads/monica_SIV/process_dir/data/sample/1.txt": {
  "FILE_SIZE": "26",
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "33204",
  "FILE_LAST_MODIFIED": "2023-12-21 18:34:18",
  "FILE_HASH": "6824f550e15de94013b70f2594fa8bc74ee7f45a"
},
"/home/student/Downloads/monica_SIV/process_dir/data/text": {
  "FILE_SIZE": "36",
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "33204",
  "FILE_LAST_MODIFIED": "2024-01-15 14:51:42",
  "FILE_HASH": "b9e6870427f507b057011df2b56d81f2626ccdbb"
},
"/home/student/Downloads/monica_SIV/process_dir/data/text copy": {
  "FILE_SIZE": "21",
  "FILE_USER_OWNER": "student",
  "FILE_GROUP_OWNER": "student",
  "FILE_ACCESS_RIGHTS": "33204",
  "FILE_LAST_MODIFIED": "2024-01-06 23:37:36",
  "FILE_HASH": "6841781499345f86bfb394d7c92406f89fea0df9"
},
"HASH_FUNCTION": "shai"
}
Observed Location Path      : /home/student/Downloads/monica_SIV/process_dir
Verification File Path      : /home/student/Downloads/monica_SIV/verification_db.json
Count of Examined Directories : 4
Count of Examined Files     : 3
Initiation Duration         : 0.00928807258605957 seconds
student@serverA: ~/Downloads/monica_SIV$

```

Figure 2.5: output of initialization mode

Verification mode Below is the output of the verification mode by using the command `"/siv.py -v -D process_dir/ -V verification_db.json -R verification_report.txt"`.

```

student@serverA: ~/Downloads/monica_SIV$ ./siv.py -v -D process_dir/ -V verification_db.json -R verification_report.txt
Observed Location Path      : /home/student/Downloads/monica_SIV/process_dir
Verification File Path      : /home/student/Downloads/monica_SIV/verification_db.json
Outcome File Path          : /home/student/Downloads/monica_SIV/verification_report.txt
Count of Examined Directories : 4
Count of Examined Files     : 3
Count of Warnings Issued    : 0
Verification Duration       : 0.001939535140991211 seconds

```

Figure 2.6: output of verification mode

Modification of file in process_dir which is shown in the below picture.

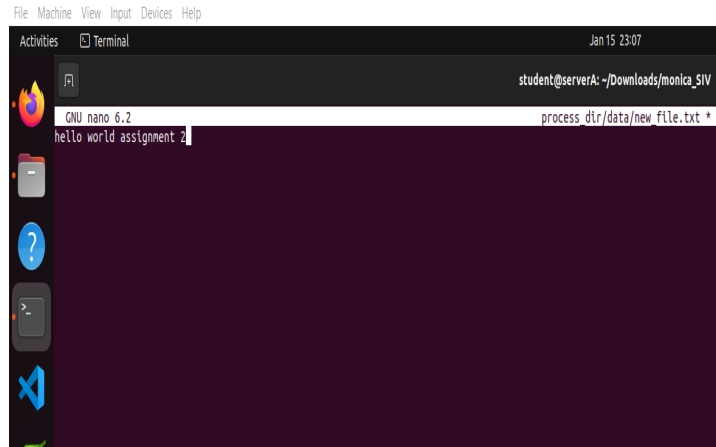


Figure 2.7: Modification of file

now again running the verification mode command we can see the last modification.



Figure 2.8: Output of verification with last modification