Assignment 2 : SIV

ET2595 Network and System Security

M Ahmad Imran Rafique
199702043515

Instructor: *Dr. Dragos Ilie & Dr. Victor Kebande*
Date: January 10, 2023

Contents

1 Environment

The Assignment was performed in following environment

1. Windows 11 host
2. Virtualbox with ET2595.ova file as the guest environment
3. On server A of the ET2595.ova guest environment
4. using pythin 3.10.6
5. connected remotely to serverA
6. using VS Code IDE

2 Introduction

2.1 Goal of the SIV

The goal of this assignment is to implement a system integrity verifier. It is used to check whether existing state of system has not been manipulated by any means other than the allowed ones. Such integrity checks are common when downloading or installing operating systems or game software.

2.2 The assignment goal

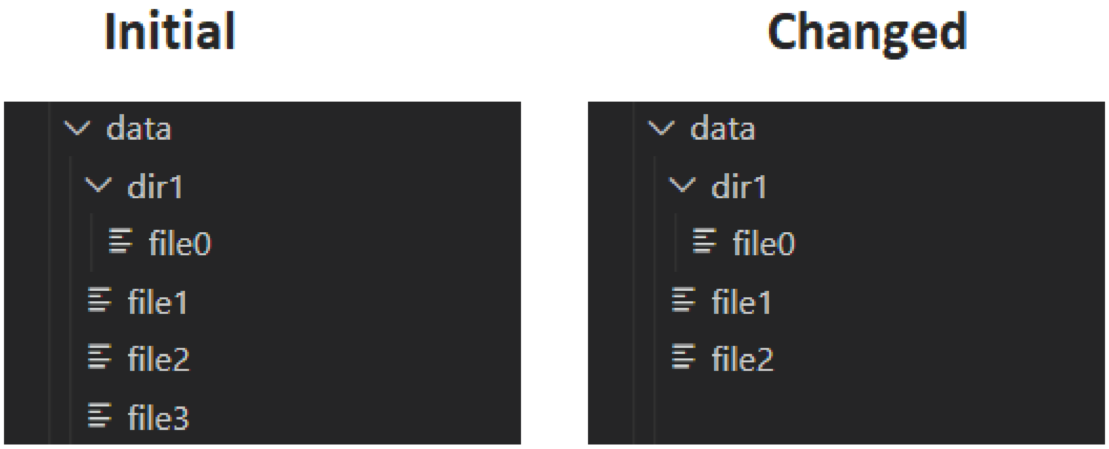
This assignment seeks to develop a system integrity verifier. The gist is that we scan or initialize a directory to be monitored where we traverse all files and folder of the directory and record information about

> **file path , size, owner, group, permission, last modified and hash**

The assignment also asks us to report in case of initialization

> **full directory path, directory and files checked, time to run, path of verification file**

And in case of verification, additionally check how many were changed and specify what changes occurred.
And record them somehow. The next part is to verify whether the integrity of the files are same as before i-e, any deletion, addition, modification of files must be reported against the initialized data we have. Consider image 1. The change of deletion must be recorded.


Initial

Changed

Figure 1: Initial Directory and Changed Directory

3 Design and Implementation

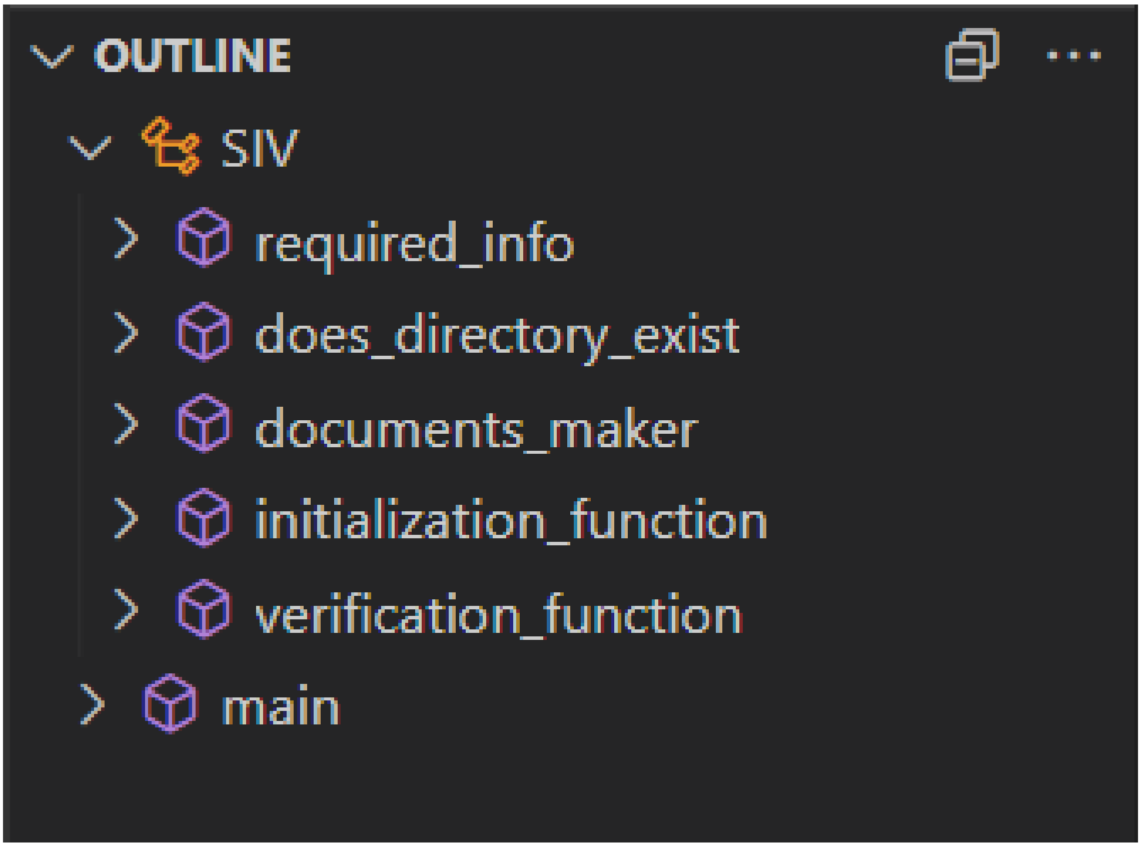
3.1 Choice of Programming Language

Python was chosen for following reasons

1. easier to work and implement code on
2. rich modules with functions
3. lightweight and less time needed to setup and run
4. The assignment was a scripting task and using plain bash seemed complicated as compared to python.

3.2 Code Structure and Flow

The code has been written in python and has the following structure



OUTLINE

SIV
- required_info
- does_directory_exist
- documents_maker
- initialization_function
- verification_function

main

Figure 2: Code Structure

The code has been written in an object oriented manner for easier looking. The code can still be improved by adding more modularity but due to time, only so much has been done.
The code can run in either initialization or verification. Following is an example

Initialize : ./siv.py -i -D data -V vDB -R init.txt -H sha1

Verify : ./siv.py -v -D data -V vDB -R verify.txt

To add, the -H also works with -H md5 since we can use either of the two for this assignment.
The code flow is as follows :

1. Enter initialize or verification mode
2. Check whether monitoring directory exists
3. Check whether we are outside monitoring directory
4. Check whether verification and report files are being made outside monitoring directory
5. For initialize, we can overwrite verification file
6. For verification, we can overwrite report file

3.3 Code Functions

The code has a class called SIV which has 5 functions a main() function. Following is a description functions other than verification and initialization.

1. required_info() : Taking file path as the input, This function collects the information of size, owner,group, permission and last modified of the file. The os, stat,time and pwd packages are needed here. The os package uses os.path.getsize() to calculate size. The os.stat() commands returns statuses like owner, group and permissions. The pwd package and stat package make it so that the outputs from the commands are human readable. For instance, os.stat() on the path can be used to get uid of the owner. We wish for a name. The uid is checked by pwd function getpwuid() against /etc/passwd entries and returns a structure from which name can be extracted. Use getpwgid() from group. The stat package also interprets the mode or permissions revealed by os.stat(). With time library, we can collect modification date information using os.path.getmtime() function and use ctime() to display in a more readable manner. All 5 of these values are returned by the function.

2. does_directory_exist(): This function checks whether the monitoring directory exist return true or false depending on status. The os.path.exists() is used to check the status of the directory. This is a sort of gate function that will stop the initialization and verification step depending on condition.

3. documents_maker(): This function performs 2 tasks (can be simplified). The first is to ensure we are outside monitoring directory (The os.getcwd()is used to check this) and so are our verification and report files. In case the directory for verification and report is inside the monitoring directory, the directory strings will be compared and if a match is found, the program is stopped and we are prompted to exit. Once task1 is clear, in task 2, we have to create/overwrite verification and report file, depending on mode of operation. The os.open() function is used to create the file with os.CREAT option to make file if not exist with 777 permission (could have set to 644 but for assignment, it looked better),

4. main(): The command line arguments and mode check operation happen here. The args object is used to create an if else statement for both initialization and verification conditions. After specified mode conditions are fulfilled, an SIV class object with appropriate function is called.

3.4 Initialization Flow Elaborated

The initialization code starts by does_directory_exist() checking whether monitoring directory exists and then by documents_maker() ensuring the required files are outside monitoring directory and then create them by names specified in assignment.

With all checks passed, we have to traverse directories. The **os** package in python is useful in this regard. There is a loop block that allows us to traverse all files and folders of a directory (all resources used will be added at the end) by ”walking” in the directory tree using the os.walk() function. The os.dir() function gives us the directory path, directory names and file names which can further be used to loop into as was done here. where directory path + file name and directory path + directory names were run on 2 loops to traverse all files.

The loop can get file by its path. The next step is extracting the information which the required_info() does. The next step is recording the information. Albeit text files were used, the easiest way to record the information per file was using json file notation. Using file path as key, we can easily record the information per file and this method will resolve a lot of headaches during verification mode.

Finally, the json is recorded to the verification file and other information is recorded to the report file (this can be overwritten during verification with additional change count and descriptions).

3.5 Verification Flow Elaborated

The conceptual flow of verification is same as initialization with difference. We run the same directory traversing loop while loading the verification data json file as keys with their values in them.

The loop this time used file path (which is the key naming convention in verification file) to check if there is such a key in verification file. If we get a hit, we can see whether any size,owner,group,permission,modification changes may have occurred. If there is no hit, that means the verification mode has detected a new file.

With the same verification file, we can use its keys since they are path and use **os.path.exists()** function to check if the files does not exist. If we get a hit, that means we have found a deleted file since last initialization was run.

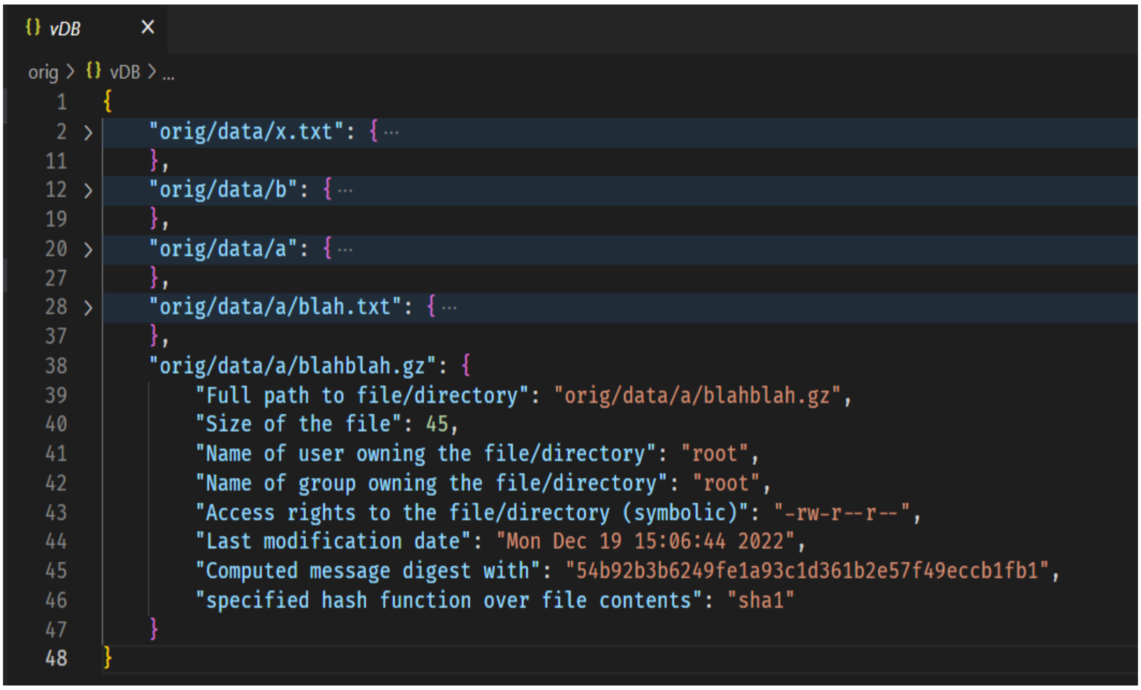
During this whole time, we will be counting all changes and adding the change descriptions to a report file.

3.6 Verification File Structure

The verification-DB file (and report but that is not relevant) as mentioned earlier uses a json syntax. The first keys is the path to directory and its value is another level of json with keys being statements of requirements extracted from the assignment descriptor and values were what was extracted by the required_info() function.

```
{
    ”path to file”: {
        ”Full path to file/directory”: <path to file>,
        ”Size of the file”: <size>,
        ”Name of user owning the file/directory”: <owner>,
        ”Name of group owning the file/directory”: <group>,
        ”Access rights to the file/directory (symbolic)”: <permission>,
        ”Last modification date”: <modification date>,
        ”Computed message digest with”: <hash value>,
        ”specified hash function over file contents”: <hash method>
    },
    ”path to file”{...
}
```

Following is a working sample.



```
{
    "orig/data/x.txt": {···
    },
    "orig/data/b": {···
    },
    "orig/data/a": {···
    },
    "orig/data/a/blah.txt": {···
    },
    "orig/data/a/blahblah.gz": {
        "Full path to file/directory": "orig/data/a/blahblah.gz",
        "Size of the file": 45,
        "Name of user owning the file/directory": "root",
        "Name of group owning the file/directory": "root",
        "Access rights to the file/directory (symbolic)": "-rw-r--r--",
        "Last modification date": "Mon Dec 19 15:06:44 2022",
        "Computed message digest with": "54b92b3b6249fe1a93c1d361b2e57f49eccb1fb1",
        "specified hash function over file contents": "sha1"
    }
}
```

Figure 3: Test code run

The idea behind using a json file was that it was easier to use a semi structured text when comparing against values in verification. And since our initial key for each block is the path itself, is much simpler to go down the block for further comparisons. In effect, its like a python dictionary, however since our program can be run in one of two modes, a compatible structure was needed to hold the data and thus json became the best candidate.

4 Usage

The usage of the script wil be shown against a test script and against a directory inside the latest linux kernel.

4.1 Test using test_siv.py

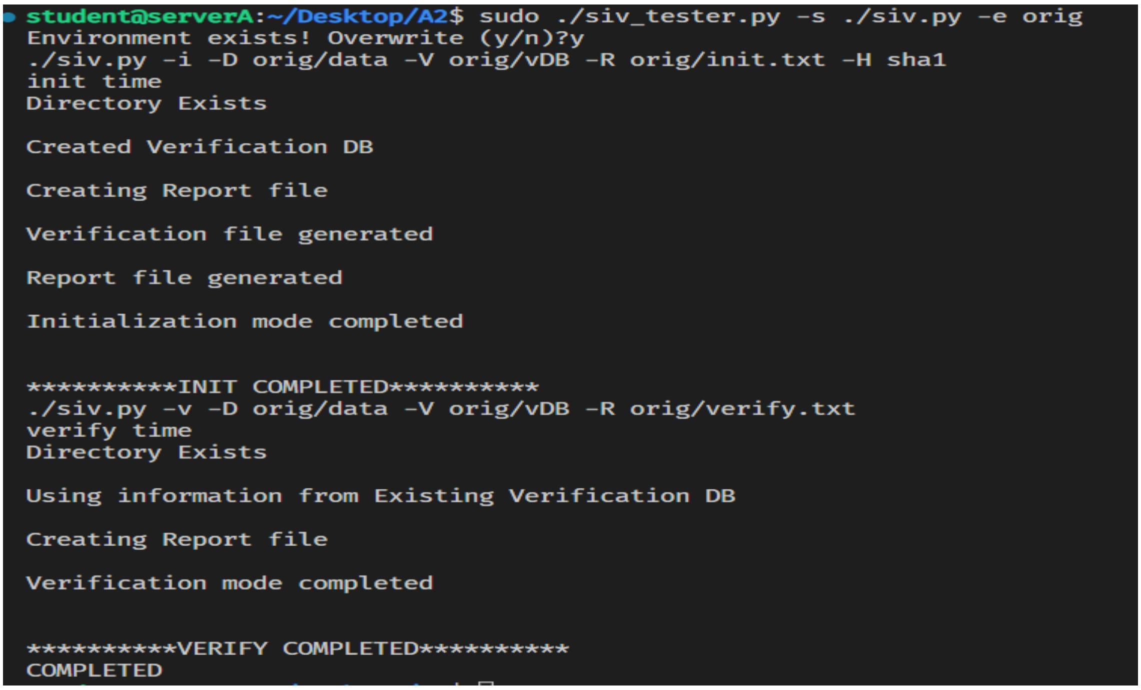
To test the code out, the test_siv.py code by Dragos Ilie to run the environment. The command line call was as follows

```
sudo ./siv_tester.py -s ./siv.py -e orig
```

This command line function should

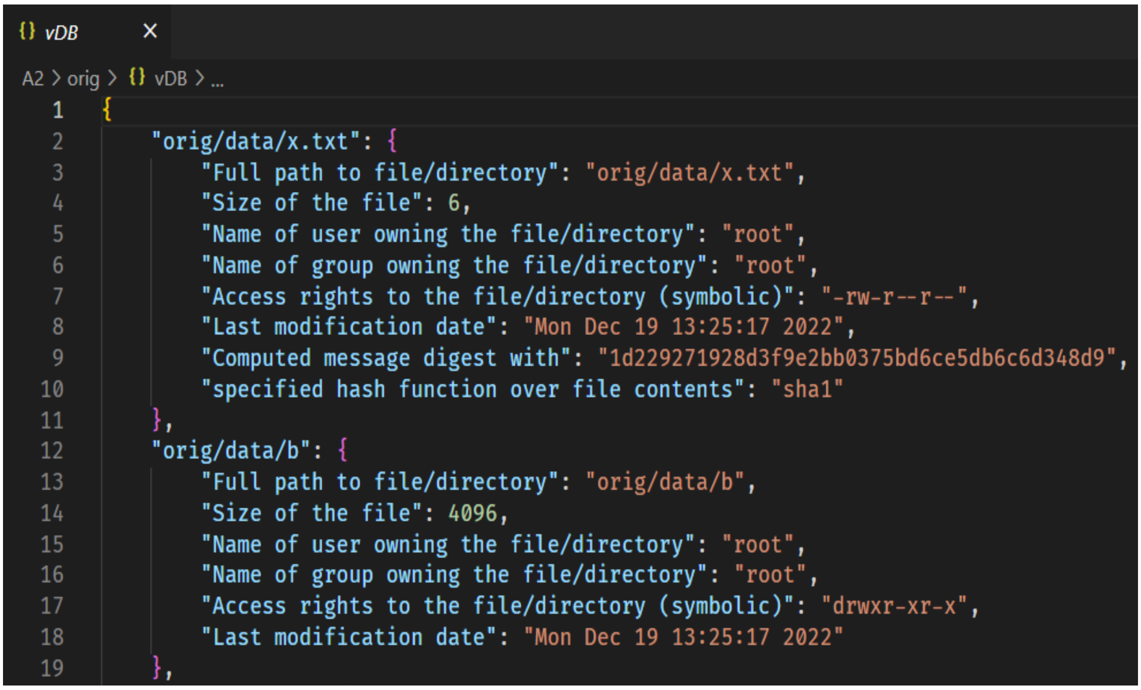
1. create or overwrite the directory by name orig with a data folder
2. perform initialization and verification and put verificationDB and report files inside orig (orig is not our monitoring directory, orig/data is)

Following is the CLI response of the execution



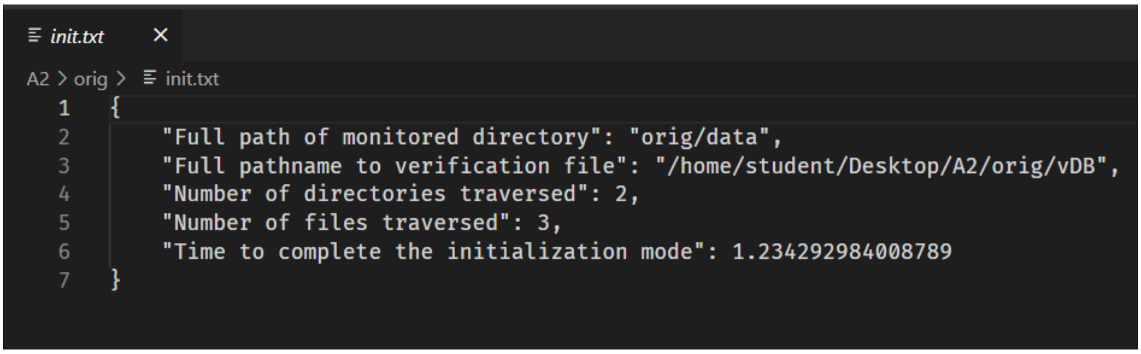
```
student@serverA:~/Desktop/A2$ sudo ./siv_tester.py -s ./siv.py -e orig
Environment exists! Overwrite (y/n)?y
./siv.py -i -D orig/data -V orig/vDB -R orig/init.txt -H sha1
init time
Directory Exists

Created Verification DB

Creating Report file

Verification file generated

Report file generated

Initialization mode completed


**********INIT COMPLETED**********
./siv.py -v -D orig/data -V orig/vDB -R orig/verify.txt
verify time
Directory Exists

Using information from Existing Verification DB

Creating Report file

Verification mode completed


**********VERIFY COMPLETED**********
COMPLETED
```

Figure 4: Test code run



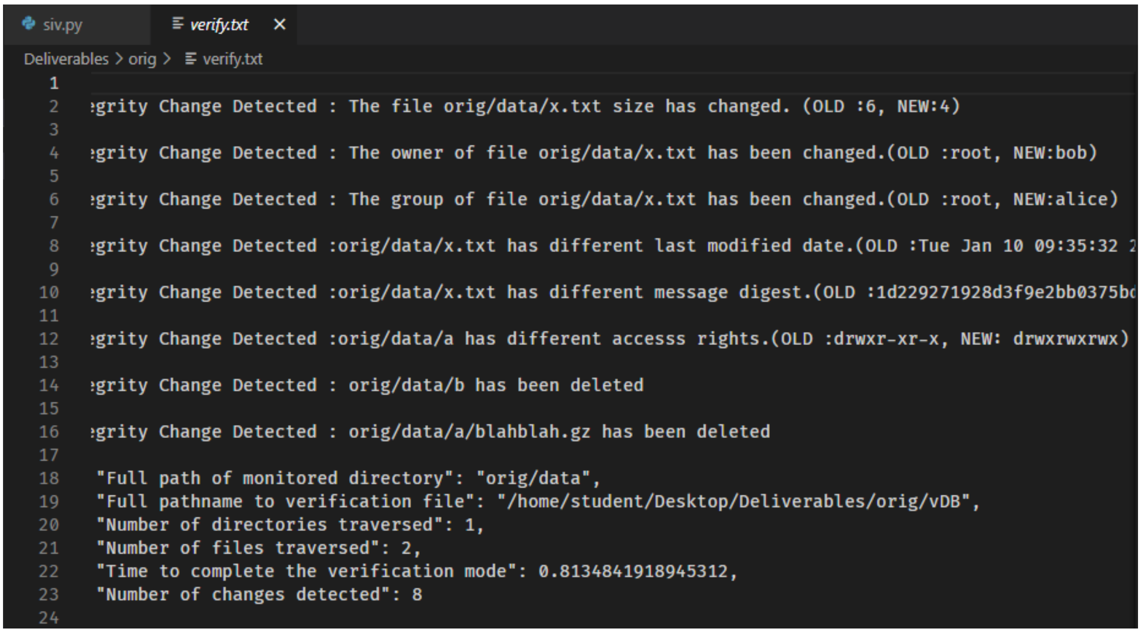
```
{
    "orig/data/x.txt": {
        "Full path to file/directory": "orig/data/x.txt",
        "Size of the file": 6,
        "Name of user owning the file/directory": "root",
        "Name of group owning the file/directory": "root",
        "Access rights to the file/directory (symbolic)": "-rw-r--r--",
        "Last modification date": "Mon Dec 19 13:25:17 2022",
        "Computed message digest with": "1d229271928d3f9e2bb0375bd6ce5db6c6d348d9",
        "specified hash function over file contents": "sha1"
    },
    "orig/data/b": {
        "Full path to file/directory": "orig/data/b",
        "Size of the file": 4096,
        "Name of user owning the file/directory": "root",
        "Name of group owning the file/directory": "root",
        "Access rights to the file/directory (symbolic)": "drwxr-xr-x",
        "Last modification date": "Mon Dec 19 13:25:17 2022"
    },
```

Figure 5: The Verification file



```
{
    "Full path of monitored directory": "orig/data",
    "Full pathname to verification file": "/home/student/Desktop/A2/orig/vDB",
    "Number of directories traversed": 2,
    "Number of files traversed": 3,
    "Time to complete the initialization mode": 1.234292984008789
}
```

Figure 6: The Initialization Report

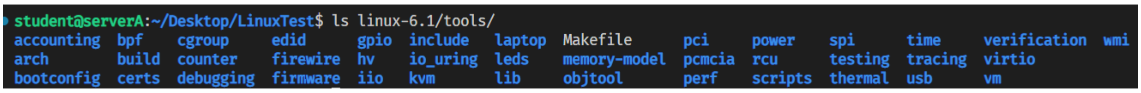


```

egrity Change Detected : The file orig/data/x.txt size has changed. (OLD :6, NEW:4)

egrity Change Detected : The owner of file orig/data/x.txt has been changed.(OLD :root, NEW:bob)

egrity Change Detected : The group of file orig/data/x.txt has been changed.(OLD :root, NEW:alice)

egrity Change Detected :orig/data/x.txt has different last modified date.(OLD :Tue Jan 10 09:35:32 2

egrity Change Detected :orig/data/x.txt has different message digest.(OLD :1d229271928d3f9e2bb0375bd

egrity Change Detected :orig/data/a has different accesss rights.(OLD :drwxr-xr-x, NEW: drwxrwxrwx)

egrity Change Detected : orig/data/b has been deleted

egrity Change Detected : orig/data/a/blahblah.gz has been deleted

 "Full path of monitored directory": "orig/data",
 "Full pathname to verification file": "/home/student/Desktop/Deliverables/orig/vDB",
 "Number of directories traversed": 1,
 "Number of files traversed": 2,
 "Time to complete the verification mode": 0.8134841918945312,
 "Number of changes detected": 8

```

Figure 7: The Verification Report

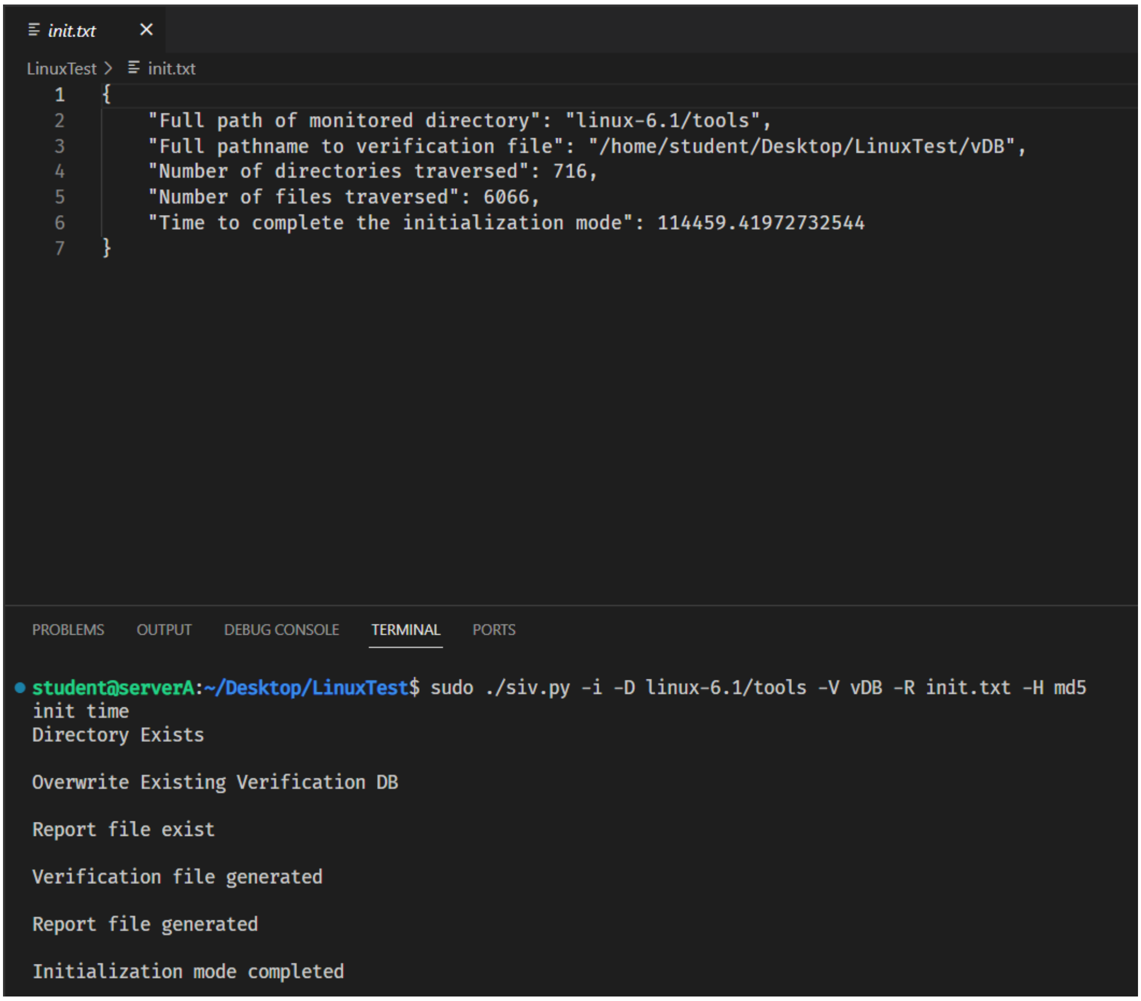
4.2 Test on the Linux Kernel

To perform a larger test, The linux kernel mainline 6.1 has been downloaded. Now, the issue is that it takes an insane amount of time to run through the code given how it is structured and then to compute the hashes. Adding more RAM did not help much so The next best idea was to select a large folder inside the kernel folder downloaded earlier. The choice was the **tools** folder.



```
student@serverA:~/Desktop/LinuxTest$ ls linux-6.1/tools/
accounting  bpf    cgroup     edid      gpio  include   laptop  Makefile      pci     power    spi      time     verification  wmi
arch        build  counter    firewire  hv    io_uring  leds    memory-model  pcmcia  rcu      testing  tracing  virtio
bootconfig  certs  debugging  firmware  iio   kvm       lib     objtool       perf    scripts  thermal  usb      vm
```

Figure 8: The tools folder



```
{
    "Full path of monitored directory": "linux-6.1/tools",
    "Full pathname to verification file": "/home/student/Desktop/LinuxTest/vDB",
    "Number of directories traversed": 716,
    "Number of files traversed": 6066,
    "Time to complete the initialization mode": 114459.41972732544
}
```

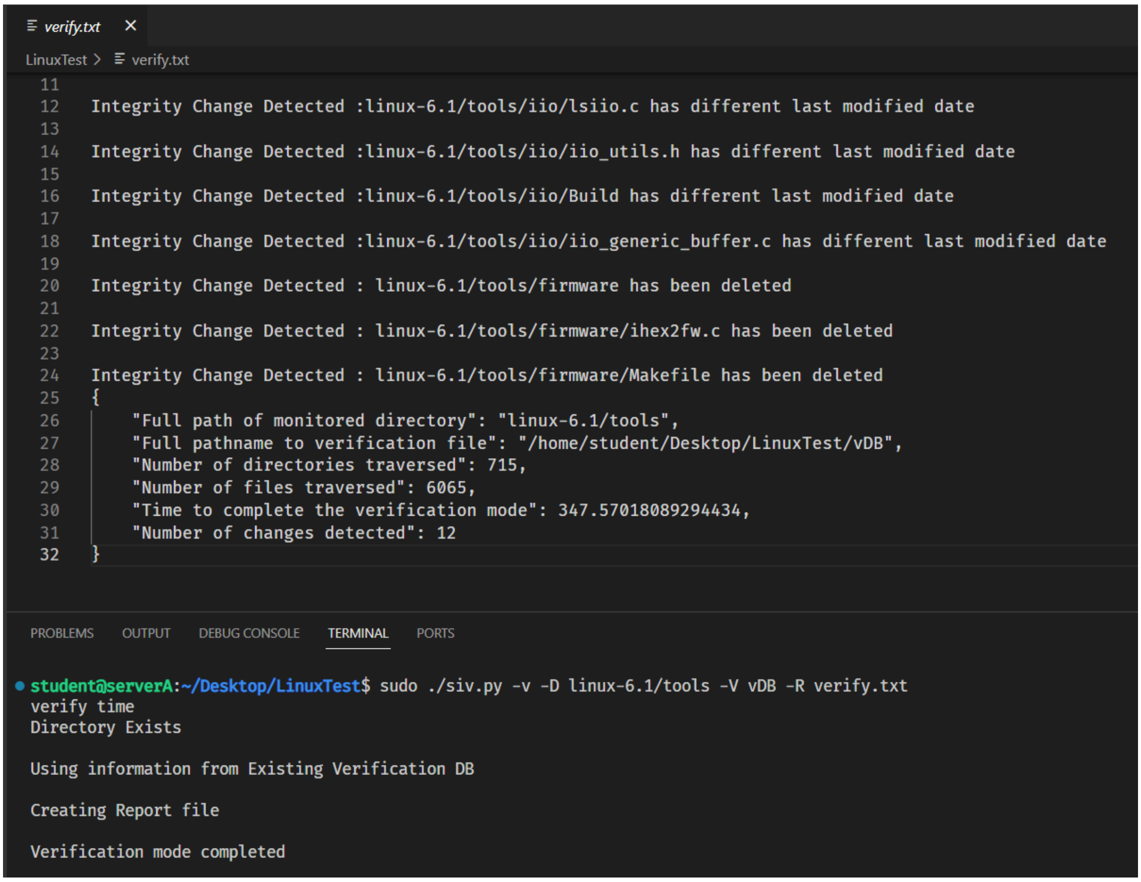
```
student@serverA:~/Desktop/LinuxTest$ sudo ./siv.py -i -D linux-6.1/tools -V vDB -R init.txt -H md5
init time
Directory Exists

Overwrite Existing Verification DB

Report file exist

Verification file generated

Report file generated

Initialization mode completed
```

Figure 9: The tools folder initialized (on an earlier version of the code)

So, we have 716 directories and a total of 6066 files inside this directory. We made the following changes inside the tools folder.

1. Touched all files inside tools/iio directory (8 files)
2. Deleted tools/firmware folder with 2 files (1 folder of 2 files)
3. Added a file inside tools/counter called yeet

In total, 12 recorded changes which after running the code in verification mode, was shown



```

Integrity Change Detected :linux-6.1/tools/iio/lsiio.c has different last modified date

Integrity Change Detected :linux-6.1/tools/iio/iio_utils.h has different last modified date

Integrity Change Detected :linux-6.1/tools/iio/Build has different last modified date

Integrity Change Detected :linux-6.1/tools/iio/iio_generic_buffer.c has different last modified date

Integrity Change Detected : linux-6.1/tools/firmware has been deleted

Integrity Change Detected : linux-6.1/tools/firmware/ihex2fw.c has been deleted

Integrity Change Detected : linux-6.1/tools/firmware/Makefile has been deleted
{
    "Full path of monitored directory": "linux-6.1/tools",
    "Full pathname to verification file": "/home/student/Desktop/LinuxTest/vDB",
    "Number of directories traversed": 715,
    "Number of files traversed": 6065,
    "Time to complete the verification mode": 347.57018089294434,
    "Number of changes detected": 12
}
```

```
student@serverA:~/Desktop/LinuxTest$ sudo ./siv.py -v -D linux-6.1/tools -V vDB -R verify.txt
verify time
Directory Exists

Using information from Existing Verification DB

Creating Report file

Verification mode completed
```

Figure 10: The tools folder verified(on an earlier version of the code)

We have 715 directories (1 folder deleted) and 6065 files (2 files removed, 1 added) and total 12 changes detected.

5 Limitations

After many runs, the implementation has following limitations

1. The verification and initialization file location ensurer is a bit faulty. To ensure the files are not made in the monitoring directory, as per my understanding, was that it is not allowed for the file to exist there. An idea was to make the file and use os package to check existence but that is just a waste of memory. The alternative was to check if the file was being made like **monitoredDirectory/verificationFile** manner. If we specify the monitoring directory as monitoredDirectory, we can perform a string split on verification or report input. However, this has a complexity issue that I was unable to code in such a way that all cases were covered. To elaborate

    ```
    -D monitoredFolder
    -V monitoredFolder/init.txt
    ```

    Is easy to split as we can check monitoredFolder in [monitoredFolder,init.txt], however

    ```
    -D /home/student/monitoredFolder
    -V monitoredFolder/init.txt
    ```

    Adds to difficulty as we now have [home,student,monitoredFolder] in [monitoredFolder,init.txt]. We can split based on last element of the first array and first of the second array but adding /home/student to even the -V argument, makes a problem, which solution was making the already spaghetti code soggier. Its entirely possible I misread the statement and the other method is allowed but as of now, it has not been implemented in a more refined way and the best expectation is that the siv script is at the same level as the monitored directory.

6 Helpful resources

The following sources were extremely helpful in creation of this assignment
For arguments parsing

1. https://docs.python.org/3/library/argparse.html
2. https://towardsdatascience.com/a-simple-guide-to-command-line-arguments-with-argparse-6824c30ab1c3
3. https://github.com/dragos-bth/siv_tester/blob/master/test_siv.py

For directory traversing

1. https://stackoverflow.com/questions/10377998/how-can-i-iterate-over-files-in-a-given-directory
2. https://stackoverflow.com/questions/927866/how-to-get-the-owner-and-group-of-a-folder-with-python-on-a-linux-machine
3. https://docs.python.org/3/library/stat.html
4. https://www.geeksforgeeks.org/python-os-stat-method/
5. https://stackoverflow.com/questions/16953842/using-os-walk-to-recursively-traverse-directories-in-python

For json handling

1. https://www.w3schools.com/python/python_json.asp
2. https://www.geeksforgeeks.org/how-to-compare-json-objects-regardless-of-order-in-python/
3. https://www.codespeedy.com/check-if-a-key-exists-in-a-json-string-or-not-in-python/
4. https://pynative.com/python-check-if-key-exists-in-json-and-iterate-the-json-array/
5. https://dev.to/bluepaperbirds/get-all-keys-and-values-from-json-object-in-python-1b2d

For file handling

1. https://stackoverflow.com/questions/35818124/using-with-open-as-file-method-how-to-write-more-than-once
2. https://stackoverflow.com/questions/10946134/in-python-how-can-i-open-a-file-and-read-it-on-one-line-and-still-be-able-to-c
3. https://cmdlinetips.com/2016/01/opening-a-file-in-python-using-with-statement/

For hash buffers

1. https://www.programiz.com/python-programming/examples/hash-file