National College of Ireland

Configuration Manual

MSc Academic Internship
MSc Cyber Security

Kedar Wattamwar
Student ID: 22116532

School of Computing
National College of Ireland

Supervisor: Vikas Sahani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Kedar Sunil Wattamwar……

Student ID: …22116532……

Programme: …MSc Academic Internship…… **Year:** ……2023…

Module: …MSc Academic Internship……..

Lecturer: …Mr. Vikas Sahani……

Submission Due Date: …14-12-2023.....

Project Title: Optimizing File Integrity Monitoring Using YARA rules

Word Count: ……1778…… **Page Count:** ……10……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……14-12-2023……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kedar Sunil Wattamwar
Student ID: 22116532

1 Introduction

This handbook includes technical specifications and instructions for installing and using the system that was developed during the academic research outlined in the primary thesis document. The next sections of this document contain information about the software used and its related versions, the development environment (hardware and software features), initial installation instructions, proper development operation, and a quick summary of the application's results. Information on the architectural and design phase can be found in the main document.

2 System Configuration

In the section, all of the software tools, frameworks, and solutions that were utilised during the development and testing of the application are included, along with a brief description of the hardware specs of the device used.

2.1 Hardware Configuration

Table 1 lists the specifications of the system's hardware and features that were used to execute the software created for the relevant topic.

Hardware	Confiuration
Processor	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
Memory	16.0 GB (15.9 GB usable)
Storage	512 GB SSD

2.2 Software Configuration

All of the information related to the software components and their details (including version and brief description) that were used in creating a new solution to the previously identified problem is contained in Table 2.

Software	Description	Version
Operating System	Windows 10 Pro	22H2
Python	The primary programming language used in the application's development(Shell script).	Python 3.11.6
Visual studio code	Microsoft source-code editor created for Linux, macOS, and Windows. Support for debugging, snippets, code and rewriting.	Version 1.84

2.3 Python Libraries

Libraries Used	Description
Hashlib	Provides a common interface to many secure hash and message digest algorithms, in this application it is used for MD5 Hash creation.
OS	Provides a way of interacting with the operating system, essential for file and directory manipulation. In this application it is used to run yara commands
Time	Allows for time-related functionality. In this application, it can be used for introducing delays or timestamps

3 System Implementation Reference manual.

The set of steps in the following section is to demonstrate how to install, configure, and start all necessary components in order to replicate the results of the system's testing phase. The areas listed below are each for a different part of the overall plan for the launch and verification of the designed application's proper functionality.

3.1 Prerequisites

Before beginning the actual code processing, an operating system has to have the necessary software components installed in order for it to communicate with the provided code:

- The application was tested using Visual Studio Code. Download and install any IDE that supports the Python programming language (Pycharm, anaconda with spider etc).
- Install Python 2 (make sure to install the version specified in the Table 2) after downloading it.

3.2 Setting up and developing the system.

After installing each item from the list above, the produced programme can be opened to extract the package containing all the necessary code. Figure 1 displays the contents of the downloaded package that was included with this manual.

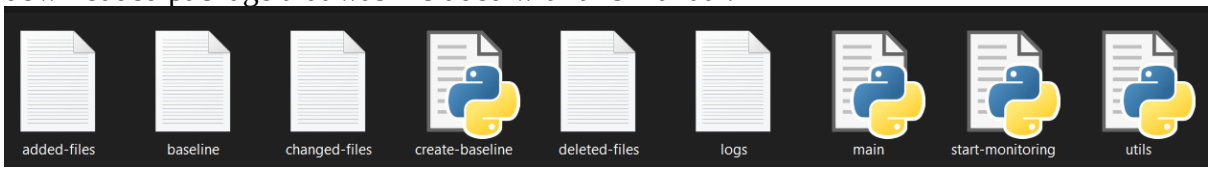

Figure 1 Contents of Zip file.

The thesis project folder contains the application that will be used for testing. The programme consists of python scripts for create baselines, utils and monitoring. The project should be extracted and can be located anywhere in the system. Making sure it is in safer environment as this is the programme which will be monitoring and giving alerts as well as creating baseline.

To execute the programme for monitoring and alerting, all the file paths must be defined in the code, this will ensure that the hash values as well as logs are saved properly for future references.

3.3 An Overview of the Source Code for the Application

In addition to describing the modules of the application that correlate to various activities that will occur when the application is running, the following paragraph will show how to launch the developed application after opening it in the IDE. First step is to create the base line, as the code keeps the hash of these files, the hashlib library is imported. Set the path variable in the script to the absolute path of the directory containing the filest hat are needed to be monitored. As the code takes file path as well as hash of the files, it is important to provide correct file path to the code.

```
2   import hashlib
3
4   path = 'C:/Users/Hp/Desktop/fim/files'
```

Figure 2: Imports and File path.

```
6   def getFiles():
7       return os.listdir(path)
8
9   def getFilesPath():
10      return path
11
12  def createHashAppendFilePath(filename):
13      BUF_SIZE = 32768 # Read file in 32kb chunks
14      SHA256 = hashlib.sha256()
15      with open(filename, 'rb') as f:
16
17          data = f.read(BUF_SIZE)
18          SHA256.update(data)
19          return SHA256.hexdigest() + "|" + filename + "\n"
20
21  def hashFiles():
22      filesList = getFiles()
23      f = open("baseline.txt", "w")
24      for x in range(0, len(filesList)):
25          print(filesList[x])
26          f.write(createHashAppendFilePath(path + "/" + filesList[x]))
27      f.close()
28      print("Baseline Created Succesfully.")
29
30
31  # import hashlib
32
33  def sha256sum(filename):
34      with open(filename, 'rb', buffering=0) as f:
35          return hashlib.file_digest(f, 'sha256').hexdigest()
```

Figure 3: Functionalities.

There are additional functionalities like getFiles() which gets a list of file names in the specified directory and getFilesPath() that retrieves the path of the directory containing the files. When this method of creating baseline is called, it takes file hashes and saves it in a text file named baseline.

Once baseline is created, the monitoring process can be started. The monitoring process has six phases of monitoring, initialization, monitoring file addition, monitoring file deletions, file modifications and for continuous monitoring. with additional subphase in file addition for scanning the malicious content in the added file using Yara rule. In order for monitoring and

running CLI commands libraries like hashlib, time and OS were imported with methods from utils like getFiles, getFilesPath, hashFiles, and createHashAppendFilePath were also imported.

3.3.1 Initialization Phase:

- This phase initializes the script by reading baseline information and creating dictionaries/lists for efficient comparison.
- Opens and reads the baseline.txt, added-files.txt, deletefiles.txt, and changed-files.txt files to get information about the baseline, newly added files, deleted files, and changed files, respectively.
- Gets the list of all files in the specified directory using the getFiles() function.
- Initializes various lists and dictionaries to store file information and hashes for efficient comparison.

```
6
7   def startMonitoring():
8       baselineFile = open('baseline.txt', 'r')
9       linesInBaseline = baselineFile.readlines()
10
11      newFilesBaseline = open('added-files.txt', 'r')
12      linesInNewFilesBaseline = newFilesBaseline.readlines()
13      newFilesList = []
14      newFilesHashes = []
15      deletedFile = open('deleted-files.txt', 'r')
16      linesInDeletedFiles = deletedFile.readlines()
17      deletedFilesList = []
18
19      files = getFiles()
20      path = getFilesPath()
21      baselineFilesList = []
22      baselineFileHashes = []
23      currentDirectoryFiles = []
24      currentDirectoryFilesHashes = []
25      changedFile = open('changed-files.txt', 'r')
26      changedFilesListLines = changedFile.readlines()
27      changedFilesListHashes = []
28      currentDirectoryFilesDictionary = {}
29      baselineFilesDictionary = {}
```

Figure 4: Initialization process.

3.3.2 Monitoring File Additions:

- Goes through the list of current files in the directory.
- Checks if a file is not in the baseline and not in the list of newly added files.
- Logs the addition of the new file in logs.txt.
- Updates added-files.txt with the hash of the new file.
- Starts a Yara scan on the newly added file using the specified Yara rule.

```
54
55      for i in range(0, len(currentDirectoryFiles)):
56          if((currentDirectoryFiles[i] not in baselineFilesList) and (currentDirectoryFiles[i] not in newFilesList)):
57              logFile = open('logs.txt', 'a')
58              logFile.write("NEW FILE ADDED: " + currentDirectoryFiles[i] + "\n")
59              logFile.close()
60              print("New File Added!: ", currentDirectoryFiles[i])
61              addedFile = open('added-files.txt', 'a')
62              newFileHash = createHashAppendFilePath(currentDirectoryFiles[i])
63              addedFile.write(newFileHash)
64              addedFile.close()
65              
66              yara_command = (
67                  r"C:\Users\Hp\Desktop\fim\files\yara64 -s -r "
68                  r"C:\Users\Hp\Desktop\fim\files\yararule.yara " + currentDirectoryFiles[i]
69              )
70              os.system(yara_command)
71
```

Figure 5: Monitoring file additions.

3.3.3 Monitoring File Deletions:

- Goes through the list of baseline files.
- Checks if a baseline text file is not in the current directory and not in the list of deleted files.
- Logs the deletion of the file in logs.txt.
- Updates deleted-files.txt with the path of the deleted file.

```
71
72      for i in range(0, len(baselineFilesList)):
73          if((baselineFilesList[i] not in currentDirectoryFiles) and (baselineFilesList[i] not in deletedFilesList)):
74              print("FILE deleted is", baselineFilesList[i])
75              f = open('logs.txt', 'a')
76              f.write("FILE DELETED: " + baselineFilesList[i] + "\n")
77              f.close()
78              newFile = open('deleted-files.txt', 'a')
79              newFile.write(baselineFilesList[i] + '\n')
80              newFile.close()
81
```

Figure 6: Monitoring file deletions.

3.3.4 Monitoring File Modifications:

- Goes through the list of current file hashes.
- Checks if a file hash is not in the baseline file hashes and not in the list of changed file hashes.
- For each modified file, goes through the list of baseline file hashes.
- Checks if a baseline file hash is not in the current file hashes.
- Logs the change in logs.txt.
- Updates changed-files.txt with the path of the modified file.

```
82
83  for i in range(0, len(currentDirectoryFilesHashes)):
84      if((currentDirectoryFilesHashes[i] not in baselineFileHashes) and (currentDirectoryFilesHashes[i] not in changedFilesListHashes)):
85          for j in range(0, len(baselineFileHashes)):
86              if((baselineFileHashes[j] not in currentDirectoryFilesHashes)):
87                  print("File changed: ", baselineFilesDictionary.get(baselineFileHashes[j]))
88                  logFile = open('logs.txt', 'a')
89                  logFile.write("FILE CHANGED: " + baselineFilesDictionary.get(baselineFileHashes[j]) + "\n")
90                  logFile.close()
91                  changedFile = open('changed-files.txt', 'a')
92                  changedFile.write(baselineFilesDictionary.get(baselineFileHashes[j]) + '\n')
93                  changedFile.close()
94
```

Figure 7: Monitoring file modifications.

3.3.5 Continuous Monitoring:

- Initiates an infinite loop for continuous monitoring.
- Calls the startMonitoring() function in each iteration.
- To stop the script, you can manually terminate it by pressing Ctrl + C in the command prompt or terminal.

```
99
100 while(True):
101     startMonitoring()
102     #time.sleep(1)
103
```

Figure 8: Monitoring loop

4 Launching the Programme and Examining the Results

To monitor the file integrity, run the create baseline file which will create a base with hashes and paths for future comparisions. As the integrated IDE terminal will be where all text prompts and other output are shown, make sure it is visible. If an error arise during the operation of the programme, the relevant text message will show up in the console window, describing the possible source of the issue (e.g., incorrectly configured path, trouble in executing a certain command, etc.). Before starting to monitor the files, make sure that all files which capture and save logs such as addition, deletion or modifications are cleared out so no error are displayed one the monitoring process starts.

Go to stat monitoring file and run it so the monitoring process starts, make changes in the files for getting alerts and notifications a bout it. Once the files are modified, the alert will be raised with the name of the file that is changed and this will also be logged in changed file folder. Similarly if a file is deleted it will be alerted and saved in deleted files folder. For addition of any file, the alert ops up with scanning the added file with the predefinrd Yara rule, and if anything suspicious is detected it is also notified to the users. Not only that the added file hash is saved in a different file named added files, where users can copy this hash and check it on websites like virustotal for any malicious history of that file or file hash.

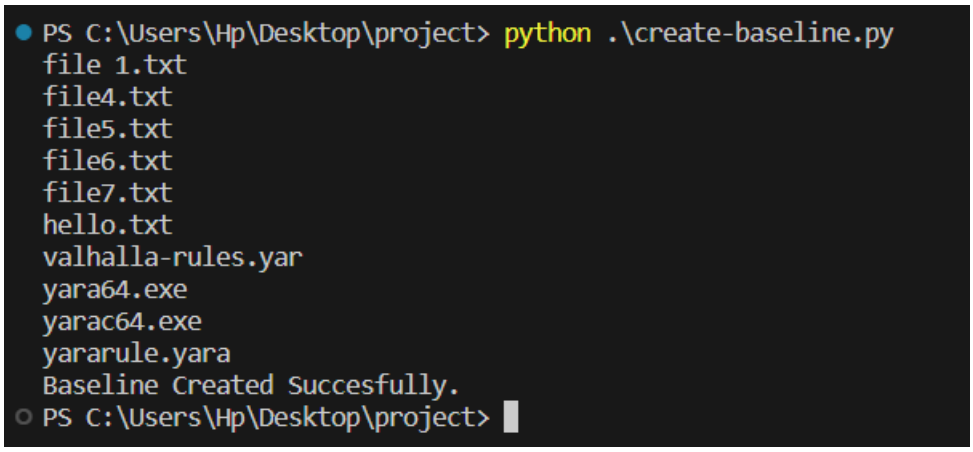

Figure 9: Baseline Creation.

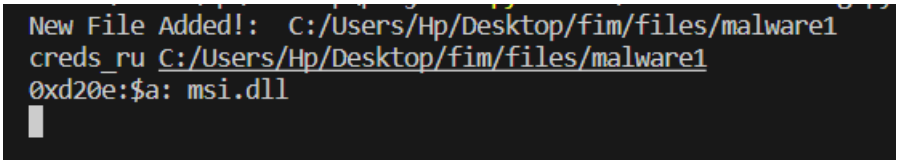

Figure 10: File addition alert with malicious content detected by Yara rule.

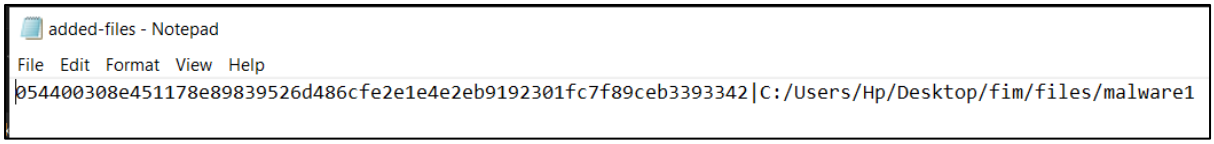

Figure 11: Added file with it SHA256 hash saved in added files folder.

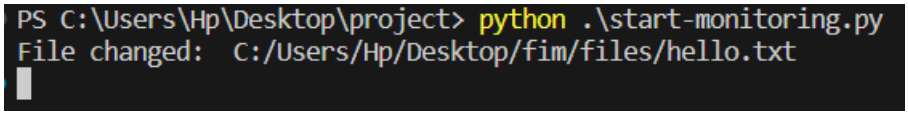

Figure 12: File modification alert.

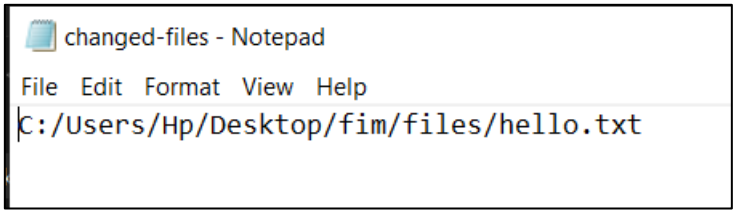

Figure 13: Trigerred Alert saved in changed files folder.

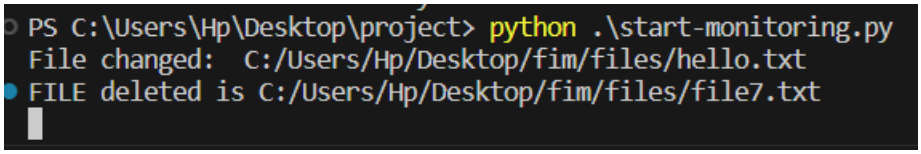

Figure 14: File deletion alert.

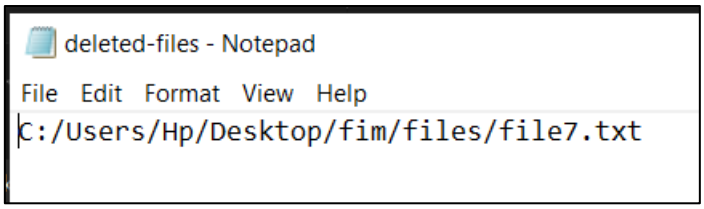

Figure 15: : Trigerred Alert saved in deleted files folder.

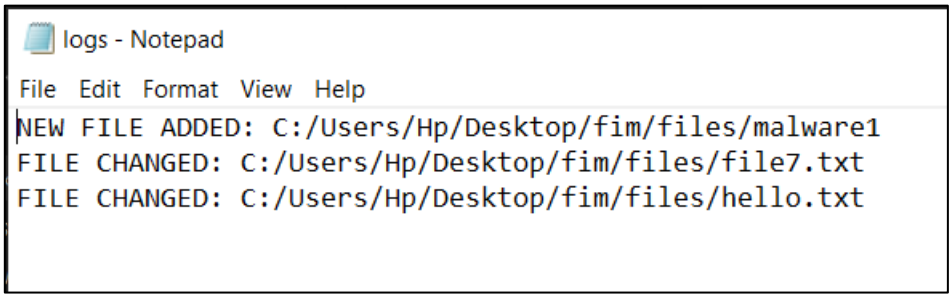

Figure 16: All the logs saved in logs folder.