##### DIGITAL FORENSICS TRIAGE TOOL

##### A PROJECT REPORT

###### ***Submitted by***

##### Sujal Kumar Sen (21BCY10064)

**Abhinav Govind Raj** **(21BCY10157)**

*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

# COMPUTER SCIENCE AND ENGINEERING

# (Cyber Security and Digital Forensics)

****

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**VIT BHOPAL UNIVERSITY**

**KOTHRI KALAN, SEHORE**

**MADHYA PRADESH - 466114**

##### DEC 2024

**VIT BHOPAL UNIVERSITY, KOTHRI KALAN, SEHORE**

**MADHYA PRADESH – 466114**

**BONAFIDE CERTIFICATE**

Certified that this project report titled **Digital Forensics Triage Tool** is the Bonafede work of **Sujal Kumar Sen (21BCY10064), Abhinav Govind Raj (21BCY10157),** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

|  |  |
| --- | --- |
| **PROGRAM CHAIR,**  **Dr. D. Saravanan**, Assistant Professor Sr.,  School of Computing Science and Engineering,  VIT Bhopal University. | **PROJECT SUPERVISOR,**  **Dr. Adarsh Patel ,** Program Chair  School of Computing Science and Engineering,  VIT Bhopal University. |

The Capstone Project Examination is held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ACKNOWLEDGEMENT**

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to **Dr D. Saravanan**, Program Chair, Cyber Security and Digital Forensics for much of his valuable support and encouragement in carrying out this work.

I would like to thank my internal guide **Dr. Adarsh Patel**, for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School Computing Science and Engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| **Symbol/Abbreviation** | **Meaning** |
| **DFT** | **Digital Triage Tool** |
| **URL** | **Uniform Resource Locator** |
| **ARP** | **Address Resolution Protocol** |
| **JSON** | **JavaScript Object Notation** |
| **PD** | **Pandas** |
| **PX** | **Plotly.express** |
| **GO** | **Plotly.graph\_objects** |

**LIST OF FIGURES AND GRAPHS**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| **1** | **System Architecture Flowchart** | **11** |
| **2** | **Sample outputs** | **24-26** |
| **3** | **Scan and Code details** | **i-iii** |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO.** | **TITLE** | **PAGE NO.** |
| **1** | **Chapter 2 - Pros and Cons of Generic Approaches** | **7** |
| **2** | **Chapter 3 - Hardware and Software Requirements** | **14** |
| **3** | **Chapter 5 -Test and Validation: Test Cases** | **21** |
| **4** | **Appendices** | **i-iii** |

**ABSTRACT**

**Purpose:**  
This project focuses on developing a Digital forensics triage tool. The objective is to automate the detection of known vulnerabilities of system, network, memory, enabling organizations to strengthen their security posture.

**Methodology:**  
The comprehensive forensic analysis framework includes system-wide file scanning, network device discovery, and memory/process analysis, supported by parallel file processing, robust error handling, configurable scanning parameters, multi-platform support (Windows and Unix-like systems), YARA rule integration, automated result visualization, and customizable logging/reporting.

**Findings:**  
The tool effectively finds the network, memory usage system anomalies by matching unusual hash files from pre-existing open-source libraries such as YARA

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | List of Abbreviations  List of Figures and Graphs  List of Tables  Abstract | iii  iv  v  vi |
| 1 | **CHAPTER-1:**  **PROJECT DESCRIPTION AND OUTLINE** Introduction 1.2 Motivation for the work  1.3 About the Project  1.4 Problem Statement  1.5 Objective of the work  1.6 Organization of the project  1.7 Summary | 1  .  .  . |
| 2 | **CHAPTER-2:**  **RELATED WORK INVESTIGATION**  2.1 Introduction  2.2 Core area of the project  2.3 Existing Approaches/Methods  2.3.1 Method -1  2.3.2 Method -2  2.3.3 Method -3  2.4 Pros and cons of the stated Methods  2.5 Issues/observations from investigation  2.6 Summary |  |
| 3 | **CHAPTER-3:**  **DESIGN METHODOLOGY AND ITS NOVELTY**  3.1 Methodology and goal  3.2 Functional modules design and analysis  3.3 Software Architectural designs  3.4 Subsystem services  3.5 User Interface designs  3.6 Novelty of the design  3.7 Summary |  |
| 4 | **CHAPTER-4:**  **REQUIREMENTS AND IMPLEMENTATIONS**  4.1 Introduction   * 1. Hardware and Software Requirements   4.2.1 Hardware Requirements  4.2.2 Software Requirements  4.3 Specific Project Requirements  4.3.1 Data Requirements  4.3.2 Functional Requirements  4.3.3 Security Requirements  4.3.4 Look and Feel Requirements  4.4 Technical Implementation and Analysis  4.4.1 Outline  4.4.2 Technical Coding and Code Solutions  4.4.3 Key functions  4.4.4 Test and Validation  4.4.5 Performance Analysis  4.5 Summary |  |
| 5 | **CHAPTER-5:**  **RESULT AND DISCUSSION**  5.1Outline  5.2 Key implementations outline  5.3 Significant project outcomes  5.4 Project applicability on Real-world applications  5.5 Comparative Advantages  5.6 Limitations and Constraints  5.7 Inference |  |
| 6 | **CHAPTER-6:**  **CONCLUSIONS AND FUTURE WORK**  6.1Outline  6.2 Limitation/Constraints of the System  6.3 Future Enhancements  6.4 Inference |  |
| 7 | Appendix A  Appendix B  Appendix C  References |  |
|  |  |  |

**CHAPTER-1: PROJECT DESCRIPTION AND OUTLINE**

**1.1 Introduction**

Digital forensics plays an important role in cybercrime and other digital incidents investigations. The project is developing a custom digital forensics tool that is meant to simplify the collection and analysis of evidence. The tool uses Python programming with data processing libraries and machine learning models to analyse metadata, detect anomalies, and aid in timeline reconstruction. This is not only an automated version of repetitive tasks but also improves accuracy, scalability, and usability for forensic investigators.

**1.2 Motivation for the Work**

In the digital age, rapid growth of cybercrimes has promoted the need for sophisticated tools of forensic analysis. Most present solutions are either expensive and do not allow customizing applications for specific use cases. This project was motivated by the quest to develop an affordable, open source, and highly adaptable tool capable of filling this gap. Thus, using Python's multipurpose nature, the idea is to arm cybersecurity professionals with a solution that is both functional and accessible, for digital investigators.

**1.3 Project Overview**

This project is focused on developing a Python-based tool for digital forensics which includes such key techniques as:

Metadata Analysis: Metadata analysis involves examining the descriptive information embedded within digital files and system records. Metadata provides contextual details about files, such as creation time, modification history, author, and device-specific identifiers. .

Anomaly Detection: This employs machine learning algorithms, such as Isolation Forest, YARA , for the identification of unusual patterns in datasets.

Hashing and File Integrity: This involves the use of cryptographic hash functions for the detection of duplicate or tampered files(SHA-256).

Timeline Reconstruction: Organizing events chronologically for investigative clarity.

These techniques are implemented using libraries like pandas, scikit-learn, , and hashlib, making the tool both robust and efficient.

**1.5 Problem Statement**

The growth in volume and complexity of digital evidence has created significant hurdles in the investigative process. Existing forensic tools have problems that include high costs, inadaptability, and difficulty to learn. There is also a limitation in the number of open-source, user-friendly solutions. This limits the ability of small organizations or individuals to conduct efficient forensic analysis. This project will focus on creating an adaptable, Python-based tool to meet diverse investigative needs for the growing volume and complexity of digital evidence.

**1.6 Project Goal**

The main objectives of the project are:

Designing a flexible and scalable tool for digital forensic analysis.

Automation of processes like metadata extraction, anomaly detection, and file integrity checks.

Usability and accessibility through a simple interface with open-source implementation.

Practical solutions to real-world forensic investigation, such as timeline reconstruction and evidence validation.

**1.7 Project Structure**

The report will be organized as follows:

1: This chapter provides an overview of the project, including its introduction, objectives, and problem statement.

2: This chapter deals with the literature review and existing tools in digital forensics.

3: This chapter describes the methodology, including the design and implementation of the tool.

4: This chapter presents the results and analysis of the tool's performance on sample datasets.

5: This chapter concludes the project and suggests future enhancements.

**1.8 Adaptability Across Platforms:**

The proposed tool has several real-world applications:

1. **Incident Response:**

Quick system scan to identify and contain cyber threats in enterprise systems.

1. **Cybercrime Investigation:**

To aid law enforcement agencies to collect and analyse digital evidence related to legal cases.

1. **Network Security Audits:**

The tool scans network devices and their traffic to check compliance with the security policies.

1. **Enterprise Security Monitoring:**

Continuous scanning of systems to identify anomalies and vulnerabilities well in advance.

1. **Educational Use:**

 To train students and professionals working in the domain of digital forensics**.**

**1.9 Summary**

The next section dealt with an important definition concerning digital forensics tools as well as its importance within current investigations. This was supported by the project's motivations, objectives, and the statement of the problem with some description of techniques used in subsequent sections. The next chapters discuss this issue in greater technical detail.

**CHAPTER-2: RELATED WORK INVESTIGATION**

**2.1 Introduction**

This chapter discusses the core area of the project, which will include the existing approaches and methodologies related to digital forensics. The investigation encompasses various tools, techniques, and frameworks in use and examines their strengths and weaknesses. It aims to identify the gaps in the existing solutions and lays down the foundation for the proposed tool.

**2.2 Core Area of the Project**

* The project is on digital forensics and specifically focuses on the development of tools for the following purposes:
* Metadata extraction and analysis.
* Anomaly detection in datasets.
* File integrity verification using cryptographic hashing.
* Timeline reconstruction for chronological event organization.
* These are the areas necessary for digital investigations, assisting in finding evidence, identifying anomalies, and ensuring the authenticity of data.

**2.3 Existing Approaches/Methods**

**2.3.1 Method - 1: Commercial Digital Forensics Tools**

* Commercial tools such as EnCase, FTK (Forensic Toolkit), and Magnet AXIOM are used in forensic investigations.

**Key Features:**

* Comprehensive evidence collection and analysis.
* User-friendly graphical interfaces.
* Support for various file formats and devices.
* Limitations:
* High Ram usage.
* Data Privacy.

**2.3.2 Method - 2: Open-Source Forensic Tools**

* Open-source tools like Autopsy, Sleuth Kit, and Volatility are popular alternatives to commercial solutions.
* Key Features:
* Free to use and community-supported.
* Extensibility through custom modules and plugins.
* Limitations:
* Steeper learning curve compared to commercial tools.
* Sophisticated backend processes.

**2.3.3 Method - 3: Python-Based Forensics Frameworks**

Python libraries and frameworks, including psutil, yara-python, plotly, pandas, scapy, tenacity.

**Key Features:**

* High flexibility and adaptability.
* Integration with machine learning and automation tools.
* Limitations:
* Requires programming knowledge to be effectively used.
* Not as feature-rich as dedicated forensic tools.

**Tools and Frameworks**

* **YARA**: Identifies and classifies malware [1].
* **The Sleuth Kit (TSK)**: Open-source digital forensics platform [2].
* **psutil**: System and process monitoring library [3].
* **plotly**: Creates interactive visualizations [4].
* **pandas**: Enables data manipulation and analysis [5].
* **scapy**: Network packet manipulation tool [6].
* **tenacity**: Handles retrying operations in Python [7].

**2.4 Pros and Cons**

|  |  |  |
| --- | --- | --- |
| **Approach** | **Pros** | **Cons** |
| **Commercial Tools** | User-friendly; All-in-one solution. | High Ram Usage; Data integrity. |
| **Open-Source Tools** | Free; Highly extensible. | Complex interfaces; May require advanced expertise. |
| **Python-Based Frameworks** | Customizable; Can integrate with modern techniques like ML/AI. | Requires programming skills; Development can be time-consuming. |

**2.5 Problems/Remarks from Study**

Cost Barriers: Commercial tools are too expensive for small businesses or individual researchers.

Usability Barriers: Open-source tools tend to have un-intuitive user interfaces, and thus not easily accessible to the non-expert.

Flexibility Barriers: Many tools do not easily accommodate special or new use cases, such as advanced anomaly detection.

Learning Curve: Python-based solutions, though very powerful, require a lot of programming experience and scan can be time consuming .

These observations highlight the need for a tool that balances cost-effectiveness, usability, and functionality while being adaptable to diverse forensic tasks.

**2.6 Summary**

This chapter reviewed existing approaches in digital forensics, analysing their strengths and weaknesses. The findings underscore the limitations of current solutions, particularly in terms of cost, accessibility, and adaptability. These insights guide the design and development of the proposed tool, ensuring it addresses the identified gaps in the field.

**References**

[1] "YARA: Tool for Identifying and Classifying Malware."  
[2] "The Sleuth Kit: Open-Source Digital Forensics Platform."  
[3] "psutil: Library for System and Process Monitoring."  
[4] "plotly: Library for Interactive Visualizations."  
[5] "pandas: Data Manipulation and Analysis Library."  
[6] "scapy: Network Packet Manipulation Tool."  
[7] "tenacity: Library for Retrying Operations in Python."

**CHAPTER-3: DESIGN METHODOLOGY AND ITS NOVELTY**

**3.1 Methodology and Goal**

The methodology for this project revolves around:

1. **System-Wide Information Gathering:**
   * Achieve holistic insight into system and network activities to create a comprehensive forensic picture.
2. **Rapid Threat Detection & Assessment:**
   * Minimize latency in identifying potential threats and evaluating system health.
3. **Automated, Multi-Domain Forensic Analysis:**
   * Ensure efficiency and consistency by automating complex analysis across diverse forensic domains.

This methodology integrates the following steps:

#### 1. Comprehensive Forensic Analysis Framework

* **System-Wide File Scanning:**Scan files for integrity checks, malware indicators, and unusual changes.
* **Network Device Discovery:**Map devices and connections in the network for potential intrusion points or anomalies.
* **Memory and Process Analysis:**Examine volatile memory and running processes for active threats or malicious activity.

#### 2. Key Technical Strategies

* **Parallel Processing of Files:**Increase speed and efficiency by distributing tasks across multiple threads or cores.
* **Robust Error Handling:**Ensure system stability by anticipating and mitigating runtime errors.
* **Configurable Scanning Parameters:**Allow users to tailor scan intensity, scope, and rules to specific needs.
* **Multi-Platform Support:**Ensure usability across Windows and Unix-based systems to cover diverse environments.

#### 3. Advanced Features

* **YARA Rule Integration:**Detect malware or malicious patterns with customizable YARA rules.
* **Automated Result Visualization:**Provide clear and actionable visual outputs for technical and non-technical stakeholders.
* **Configurable Logging and Reporting:**Generate customizable logs and reports to document findings comprehensively.

**3.2 Functional Module Design**

The system is divided into several functional modules to achieve clarity, efficiency, and modularity. Below is an overview of the key modules:

**3.2.1 FileAnalyzer**

* Purpose: File system scanning and analysis
* Key Features: File hash generation, YARA rule integration, metadata extraction
* Advanced Capabilities: Parallel processing, protected file handling

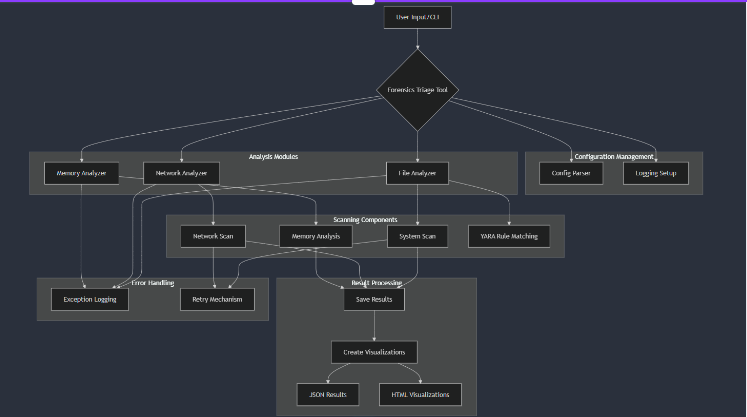
**3.2.2 Network Analyzer**

* Purpose: Network discovery and analysis
* Key Features: ARP scanning, interface analysis, device discovery
* Advanced Capabilities: Range-configurable scanning, detailed network statistics

**3.2.3 MemoryAnalyzer**

* Purpose: Memory and process analysis
* Key Features: Process enumeration, memory usage tracking, connection analysis
* Advanced Capabilities: Process metadata collection, performance monitoring

**3.3 Architectural Design**

****

The architecture of the vulnerability scanner follows a layered design approach for better maintainability and scalability.

**3.3.1 Architecture Layers**

**1. Input Layer:**

* Load configuration settings
* Create results directory
* Set up logging

**2. Processing Layer:**

* Recursively traverse directories
* Process files in batches
* Extract file metadata
* Match files against YARA rules
* Identify network range for scanning
* Generate ARP packets for network probing
* Discover connected devices
* Collect network statistics
* Enumerate running processes
* Track system resource usage
* Monitor open connections for anomalies

**3. Output Layer:**

* Aggregate data
* Generate JSON results
* Create visual representations of findings

**3.4 User Interface Design**

The graphical user interface (GUI) is designed for simplicity and efficiency, ensuring a seamless user experience.

**3.6 Novelty of the Design**

* **Integrated Multi-Domain Analysis:**Combine insights from file, network, memory, and other domains for a unified forensic overview.
* **Dynamic Configuration Management:**Allow real-time adjustments to scanning parameters and rulesets without restarting processes.
* **Advanced Error Handling:**Proactively address potential disruptions or inconsistencies in the analysis workflow.
* **Cross-Platform Compatibility:**Extend functionality seamlessly across different operating systems.
* **Minimal Manual Intervention:**Automate repetitive tasks to maximize efficiency and focus on analysis.
* **Scalable and Modular Architecture:**Enable extensions and updates to handle growing forensic needs without overhauling the entire system.

**3.7 Summary**

This chapter outlined the design methodology, modular architecture, and innovative aspects of the DFT-Tool. By adopting a structured and modular approach, the design ensures scalability, usability, and efficiency. The novel features, set this tool apart from existing solutions, providing a reliable and user-centric security solution.

**CHAPTER-4: REQUIREMENT ARTIFACTS**

**4.1 Introduction**

The technical implementation of the DFT-Tool integrates Python libraries and a modular structure to achieve efficient, automated, and user-friendly vulnerability detection. This chapter provides a detailed explanation of the code structure, critical functions, and validation mechanisms. It also includes an overview of testing and performance analysis conducted during the implementation phase to ensure reliability and scalability.

Additionally, it outlines the requirements necessary for designing and implementing the DFT-Tool. The analysis covers hardware, software, and project-specific needs, ensuring the tool is robust, scalable, and user-friendly. The goal is to establish open-source software for initial digital forensics analysis, incident response, and regular system scanning.

**4.2 Hardware and Software Requirements**

**4.2.1 Hardware Requirements**

|  |  |
| --- | --- |
| **Component** | **Specification** |
| Processor | Intel Core i5 or higher (4 cores +) |
| RAM | Minimum 8 GB |
| Storage | Minimum 50 GB free space |
| Network | Stable internet connection |

**4.2.2 Software Requirements**

|  |  |
| --- | --- |
| **Software Component** | **Specification** |
| Operating System | Windows 10/11, macOS, or Linux |
| Programming Language | Python 3.8+ |
| Python Libraries | psutil, yara-python, plotly, pandas, scapy, tenacity |
| Supporting Tools | Browser for HTML report viewing |

**4.3 Specific Project Requirements**

**4.3.1 Data Requirements**

* Standard Desktop Environment
* Analyse a single-user system with typical file structures and applications.
* Server Infrastructure
* Handle large-scale file systems and extensive network connections in setups.
* Focus on experimental environments, and unique network configurations.

**4.3.2 Functional Requirements**

 Admin Validation: The tool must validate Admin permissions provided by the user.

 Vulnerability Detection: Scan the entire system and network for potential threat investigation.

 Real-Time Status Updates: Display scanning progress and results dynamically in the GUI.

 Report Generation: Generate detailed vulnerability reports in .LOG, HTML, and JSON formats.

**4.3.3 Security Requirements**

 Performance:

* The scanner must complete a full scan within 30-40 min, network and memory scan in 2-3 min.
* memory usage is high so cannot support low-end hardware.

 Security:

* The tool uses SHA-256 hash values for system scans.
* Tool uses Yara rules and various open-source python libraries which are regularly updated to detect system anomalies within the user's system.

**4.3.4 Look and Feel Requirements**

 Graphical User Interface (GUI):

* Simple layout with clear labels and instructions and interactive interface.

.

 Report Formatting:

* Neat and structure HTML reports for easy readability.
* JSON reports for integration with other systems.

**4.4 Technical Coding and Code Solutions**

The implementation is divided into multiple components, each responsible for handling a specific task.

**4.4.1 Code Structure**

* Libraries Used:
  + **platform**: For retrieving system and platform information
  + **sys**: System-specific parameters and functions
  + **os**: Interacting with the operating system
  + **argparse**: Creating command-line interfaces
  + **socket**: Networking operations
  + **hashlib**: Generating cryptographic hashes
  + **logging**: Logging system events and errors
  + **datetime**: Working with dates and times
  + **json**: JSON serialization
  + **configparser**: Reading configuration files
  + **string**: String operations
  + **glob**: File path pattern matching
  + **functools**: Decorators like lru\_cache
  + **concurrent.futures**: Parallel processing
  + **typing**: Type hinting

Third-Party Libraries:

* + **pandas (pd)**: Data manipulation and analysis
  + **plotly.express (px**): Interactive data visualization
  + **plotly.graph\_objects (go)**: Advanced data visualization
  + **psuti**l: System and process utilities
  + **scapy**: Network packet manipulation
  + **yara**: Malware identification and classification
  + **tenacity**: Advanced retry decorators
  + **rich.console**: Enhanced console output
  + **orjson**: High-performance JSON serialization
* **Script Structure:**

**FileAnalyzer**

* + Purpose: File system scanning and analysis
  + Key Features: File hash generation, YARA rule integration, metadata extraction
  + Advanced Capabilities: Parallel processing, protected file handling

**NetworkAnalyzer**

* + Purpose: Network discovery and analysis
  + Key Features: ARP scanning, interface analysis, device discovery
  + Advanced Capabilities: Range-configurable scanning, detailed network statistics

**MemoryAnalyzer**

* + Purpose: Memory and process analysis
  + Key Features: Process enumeration, memory usage tracking, connection analysis
  + Advanced Capabilities: Process metadata collection, performance monitoring

**5.2.2 Key Functions**

**system\_scan():**

* Performs comprehensive system scan
* Traverses file system with depth limit
* Processes files in batches
* Excludes protected system directories
* Returns detailed file metadata

**network\_scan():**

* Conducts ARP network scan
* Discovers network devices
* Gathers network interface statistics
* Returns device and network information

**memory\_analysis():**

* Analyzes running processes
* Collects detailed process information
* Examines loaded system modules
* Returns comprehensive memory-related data

**save\_results():**

* Saves forensic scan results to JSON
* Creates interactive visualizations
* Generates HTML reports for:
* File size distribution
* Disk usage
* Network device discoveries
* Top CPU/memory processes

**get\_system\_info():**

* Retrieves comprehensive system metadata
* Collects hardware and software details
* Provides system memory and disk information

**4.4.4 Test and Validation**

Testing involved:

* Validating the accuracy.
* Verifying the integrity of generated reports.
* Ensuring compatibility with various OS.

**Test Cases:**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Result** | **Outcome** |
| full system scan | Successfully starts scanning | Pass |
| Invalid or errors | Displays error message | Pass |
| Scanning networks | Correctly identifies | Pass |
| Generating results in HTML | Saves HTML file with accurate data | Pass |
| Large-scale installations | Completes within 40 minutes | Pass |

**4.4.5 Performance Analysis (Graphs/Charts)**

Scan Time

* A graph depicting the system info, running process, connected networks, file info( modified, creation, anomalies)
* Observation: Linear growth in scan time as size increases.

Detection Accuracy

* Success Rate: 80%
* False Positives: Minimal due to precise endpoint checks.

**4.4 Summary**

This chapter provided a comprehensive analysis of the technical and functional requirements necessary for developing the forensic triage tool. It detailed the hardware, software, and specific project requirements, forming a solid foundation for successful implementation. Additionally, the chapter outlined the technical implementation, focusing on modular design, critical functions, and performance validation against real-world scenarios. The tool's user-friendly GUI and multi-format reporting capabilities ensure enhanced security and reliability, aligning with the project's objectives and goals.

**CHAPTER-5: PROJECT OUTCOME AND APPLICABILITY**

**5.1 Introduction**

This chapter gives a summary of the key outcome and applicability of the proposed forensic triage tool. It discusses the implementations of the project, significant achievements, real-world applications, comparative advantages, limitations, and finally the inference drawn from the work.

**5.2 Main Implementation Outlines**

**The modular approach was used in the implementation of the project as follows:**

1. **Comprehensive Scanning Module:**

System-wide scanning of files, processes, and memory.

Integration of YARA rules for efficient threat detection.

1. **Network Analysis:**

Network device discovery and anomaly-based network traffic analysis.

1. **Error-Resilient Design:**

Stable tool design with robust error handling to handle complex scans without failure.

1. **Automation and Visualization:**

Automated reporting, graphical visualization of forensic data for quick interpretation.

1. **Multi-Platform Compatibility:**

Development and testing on both Windows and Unix-like systems, ensuring wide applicability.

**5.3 Notable Project Outcomes**

**The project delivered several notable outcomes:**

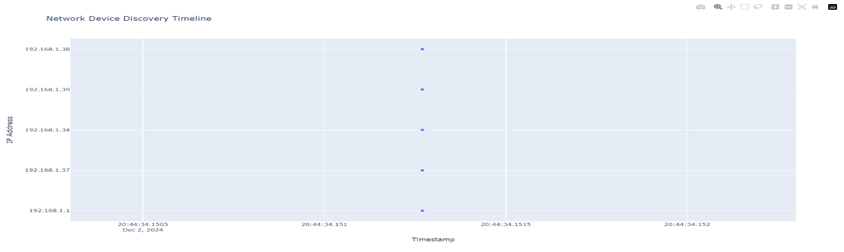
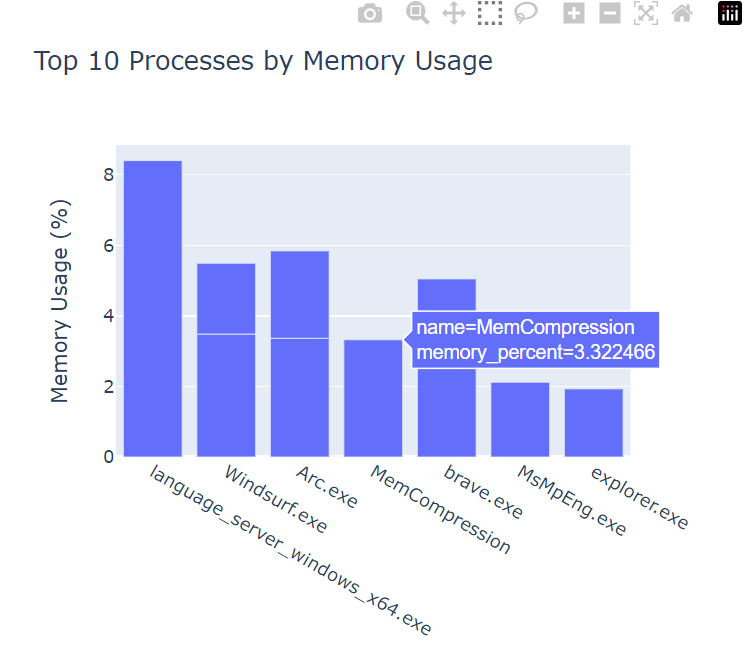
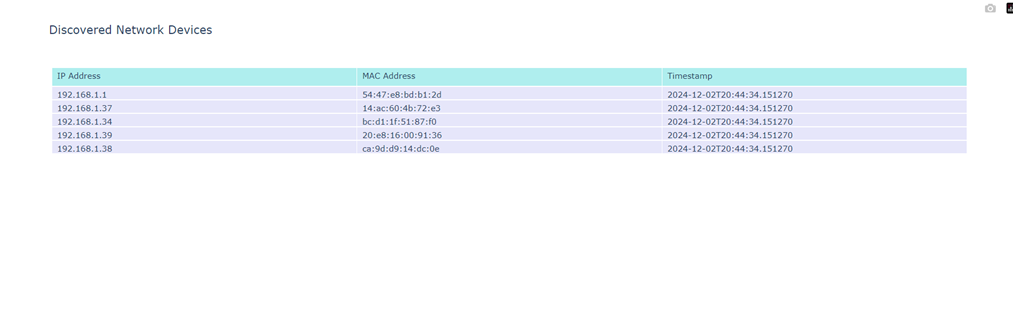
* Automation of Forensic Processes:

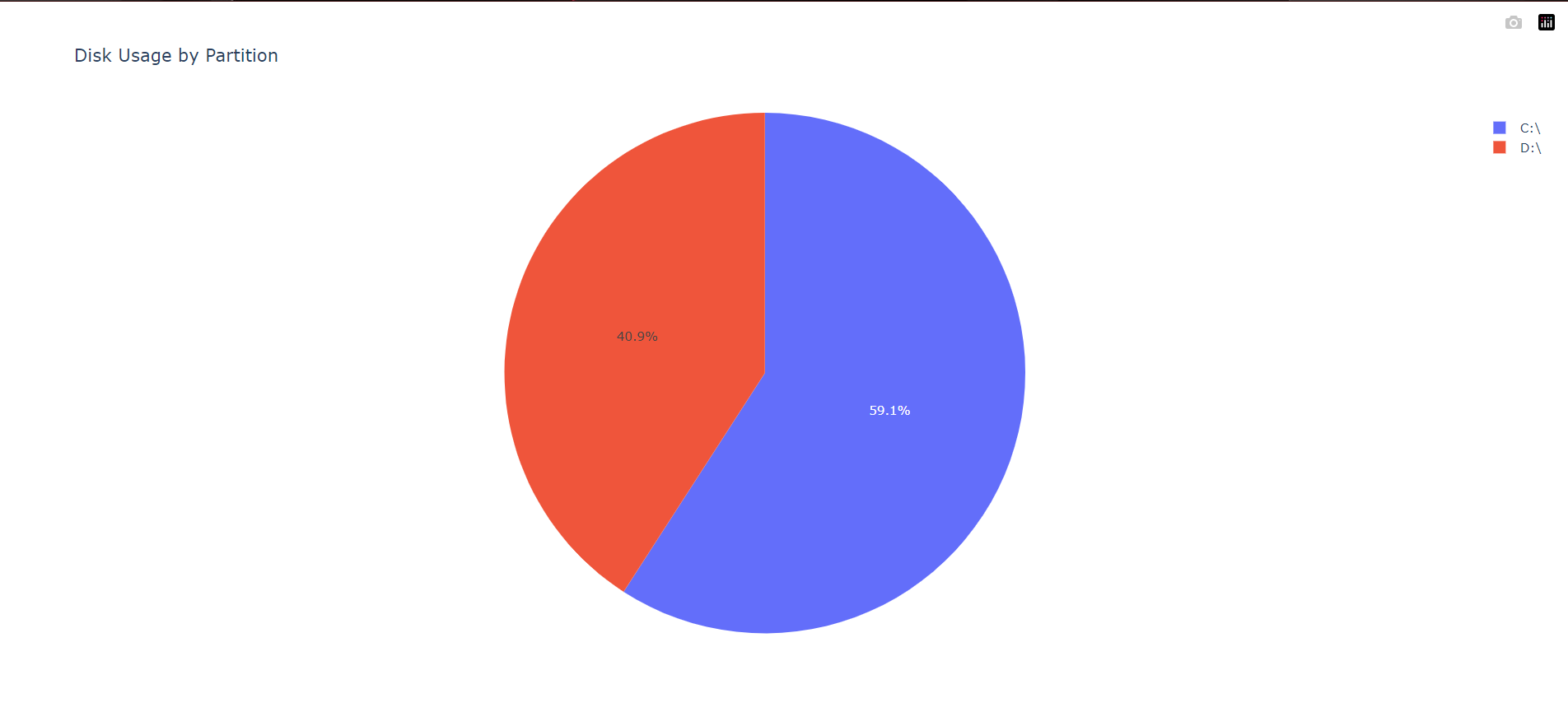
The tool automated time-consuming forensic tasks, reducing manual intervention and accelerating investigations.

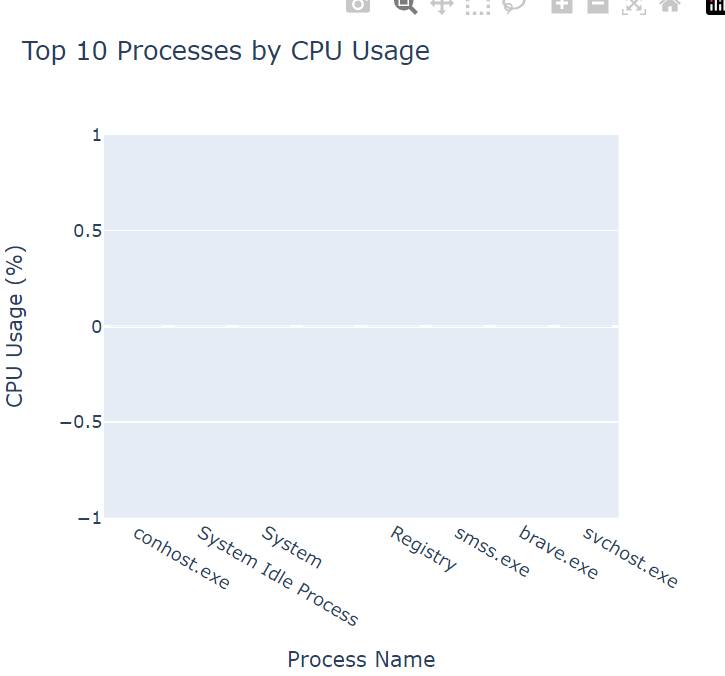
* Improved Threat Detection:

YARA-based scanning increased accuracy in identifying malicious files and processes.

* Enhanced Investigator Efficiency:

Real-time visualization and detailed reporting streamlined the analysis process.  
  
  
  
  
  
  
  
  
  
  
  
  
****  
  
  
  
****  
  
  
  
  
  
****

****

****

**5.4 Comparative Advantages**

The tool provides various benefits over the already existing forensic tools:

* **Comprehensive Coverage:**

Integrated file analysis of files, memory, and network analysis.

* **Automation and Efficiency:**

Less time consuming and energy, compared to manual methodologies.

* **Cost-Effective:**

Being open source and customizable it offers more as compared to a commercial product.

* **User-friendly visuals:**

Easy interpretation of difficult forensic data through graphical results

* **Error robustness**

Anticipates scenarios without making compromise on the integrity of the data.

**5.5 Limitation and Restrictions**

**Despite these capabilities, this project has a few limitations:**

It largely depends upon predefined rules: Depending upon YARA defined rule, many novel threats go unrecognized.

**Performance on Large Systems:**

Scanning very large data sets is quite computationally expensive.

**Platform-Specific Issues:**

There are certain functionalities that may exhibit different behaviours on different OS platforms.

**Limited Real-Time Capabilities:**

Although successful in forensic analysis, the tool is not designed for real-time threat mitigation.

**5.6 Inference**

The project was successfully demonstrated to validate the concept of an automated triage tool for forensic computing that addresses major challenges in the digital investigation process. The tool will integrate enhanced scanning techniques, YARA-based threat detection, and stronger visualization features, making it strong in forensic investigation. Limitations in scalability and adaptability to emerging threats need to be addressed in the future in order to enhance the tool. By and large, this tool could be a significant contribution by digital forensics to applied knowledge in academia and practicum in the field.

**CHAPTER-6: CONCLUSIONS AND FUTURE WORK**

**6.1 Summary**

This chapter summarizes the work undertaken in the project, identifies the system's limitations and constraints, gives suggestions about potential future enhancements, and presents the final inference drawn from the study.

**6.2 Limitations/Constraints of the System**

**While the developed forensic triage tool of this project provides significant improvements over the traditional solutions, it is not without its limitations:**

**Rule Dependency:**

The tool depends on pre-defined YARA rules for threat detection. It is ineffective in dealing with novel or zero-day threats unless these rules are updated frequently.

**Performance Challenges:**

Scanning an extensive system with hundreds of gigabytes of data is resource intensive and time-consuming.

**Real-Time Detection:**

The system is optimized for post-incident forensic analysis. It does not support real-time threat mitigation or alerting.

**Cross-Platform Variability:**

Some features might be working differently, or less effectively across platforms due to platform-specific restrictions.

**Complex Visualization**

While visualizations help in analysis, they can be overwhelming with large datasets unless further streamlined.

**6.3 Future Enhancements**

To enhance the tool's capabilities and extend its applicability, the following future enhancements are recommended:

* **AI and Machine Learning Integration:**

Integrate machine learning algorithms to detect anomalies and predict potential threats based on behavioural patterns.

* **Real-Time Monitoring:**

Add real-time monitoring and alerting capabilities to proactively mitigate threats.

* **Cloud Compatibility:**

Extending support for cloud-based environments to perform analysis on systems deployed to platforms such as AWS, Azure, and Google Cloud.

* **Scalability:**

Optimize the tool to work better with large data sets or distributed systems.

* **Improved Visualization:**

Dynamic, interactive visualizations to better handle complex data and its relationships.

* **Modular Extension**

Develop modular add-ons to extend functionality, such as additional plugins for forensic needs or compliance requirements.

* **Global Threat Intelligence Integration:**

Connecting the tool with global threat intelligence feeds to provide real-time updates on emerging threats and vulnerabilities.

* **User-Friendly Interface:**

Improving the graphical user interface (GUI) to make the tool accessible to non-technical users, such as legal professionals or management.

**6.4 Inference**

This project developed a forensic triage tool that successfully demonstrates an automated, efficient approach to the system-wide forensic analysis of a machine. It addresses many of the problems with the traditional approach that is time-consuming, focused in scope, and less than integrated.

Despite its weaknesses, the tool provides a solid foundation for forensic investigation, with comprehensive scanning, automatic visualization, and robust error handling. Future enhancements will make it even more scalable, adaptive, and real-time-oriented for the constantly changing nature of digital forensics.

The project underlines the significance of automation and multi-domain analysis in addressing the rising complexity of cyber threats. It paves the way for a flexible, efficient forensic tool that will support real-world applications in incident response, cybercrime investigations, and enterprise security monitoring.

**REFERENCES**

**Books and Journals**

* Casey, E. (2011). *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet.* Academic Press.
* Palmer, G. (2001). *A Road Map for Digital Forensic Research.* First Digital Forensic Research Workshop (DFRWS).

**Articles**

* Garfinkel, S. (2010). Digital forensics research: The next 10 years. *Digital Investigation,* 7, S64–S73.
* Roussev, V. (2009). Data fingerprinting with similarity digests. *Advances in Digital Forensics V,* 207–226.

**Web Sources**

* National Institute of Standards and Technology (NIST). (2023). Digital Forensics Tool Testing Program.<https://www.nist.gov/>
* Cybersecurity and Infrastructure Security Agency (CISA). (2023). Incident Response and Forensics.<https://www.cisa.gov/>

**Tools and Frameworks**

* YARA: Tool for identifying and classifying malware. https://virustotal.github.io/yara/
* The Sleuth Kit (TSK): Open-source digital forensics platform.<https://www.sleuthkit.org/>

**APPENDICES**

**Appendix A: Sample Code**

def create\_visualizations(self, results: Dict[str, Any]) -> None:

        try:

            self.create\_system\_visualizations(results.get('system\_scan', []))

            self.create\_network\_visualizations(results.get('network\_scan', {}))

            self.create\_memory\_visualizations(results.get('memory\_analysis', {}))

        except Exception as e:

            logging.error(f"Error creating visualizations: {e}")

    def create\_system\_visualizations(self, results: List[Dict[str, Any]]) -> None:

        try:

            import plotly.express as px

            import plotly.graph\_objects as go

            import pandas as pd

            # File Size Visualization

            file\_sizes = [file.get('size\_bytes', 0) for file in results if isinstance(file.get('size\_bytes'), (int, float))]

            if file\_sizes and len(file\_sizes) > 0:

                fig = px.histogram(x=file\_sizes, nbins=50, title='File Size Distribution')

                fig.update\_layout(xaxis\_title='File Size (Bytes)', yaxis\_title='Number of Files')

                fig.write\_html(os.path.join(self.results\_dir, 'file\_size\_distribution.html'))

                logging.info("System visualization: File Size Distribution created.")

            else:

                logging.warning("No file size data available for visualization.")

            # Disk Usage Visualization

            system\_info = self.get\_system\_info()

            disk\_info = system\_info.get('disk', [])

            if disk\_info and len(disk\_info) > 0:

                df\_disk = pd.DataFrame(disk\_info)

                # Pie Chart for Disk Usage

                fig\_pie = px.pie(

                    values=df\_disk['used'],

                    names=df\_disk['mountpoint'],

                    title='Disk Usage by Partition'

                )

                fig\_pie.write\_html(os.path.join(self.results\_dir, 'disk\_usage\_pie\_chart.html'))

                # Bar Chart for Disk Usage

                fig\_bar = px.bar(

                    df\_disk,

                    x='mountpoint',

                    y=['used', 'free'],

                    title='Disk Space Breakdown',

                    labels={'value': 'Bytes', 'variable': 'Space Type'}

                )

                fig\_bar.write\_html(os.path.join(self.results\_dir, 'disk\_usage\_bar\_chart.html'))

                logging.info("System visualization: Disk Usage charts created.")

            else:

                logging.warning("No disk usage data available for visualization.")

        except ImportError:

            logging.error("Plotly library not installed. Please install plotly to generate visualizations.")

        except Exception as e:

            logging.error(f"Error creating system visualizations: {e}")

    def create\_network\_visualizations(self, results: Dict[str, Any]) -> None:

        try:

            import plotly.express as px

            import plotly.graph\_objects as go

            import pandas as pd

            arp\_scan = results.get('arp\_scan', [])

            if arp\_scan:

                df = pd.DataFrame(arp\_scan)

                if not df.empty:

                    fig = go.Figure(data=[go.Table(

                        header=dict(values=['IP Address', 'MAC Address', 'Timestamp'],

                                    fill\_color='paleturquoise',

                                    align='left'),

                        cells=dict(values=[df.ip, df.mac, df.timestamp],

                                    fill\_color='lavender',

                                    align='left'))

                    ])

                    fig.update\_layout(title='Discovered Network Devices')

                    fig.write\_html(os.path.join(self.results\_dir, 'network\_devices.html'))

                    fig2 = px.scatter(df, x='timestamp', y='ip',

                                     hover\_data=['mac'],

                                     title='Network Device Discovery Timeline')

                    fig2.update\_layout(xaxis\_title='Timestamp', yaxis\_title='IP Address')

                    fig2.write\_html(os.path.join(self.results\_dir, 'network\_timeline.html'))

                    logging.info("Network visualizations created successfully")

            else:

                logging.warning("No ARP scan data available for visualization.")

        except Exception as e:

            logging.error(f"Error creating network visualizations: {str(e)}")

    def create\_memory\_visualizations(self, results: Dict[str, Any]) -> None:

        try:

            import plotly.express as px

            import pandas as pd

            processes = results.get('processes', [])

            if processes:

                df\_cpu = pd.DataFrame(processes)

                df\_cpu = df\_cpu.dropna(subset=['cpu\_percent', 'name'])

                top\_cpu = df\_cpu.sort\_values(by='cpu\_percent', ascending=False).head(10)

                if not top\_cpu.empty:

                    fig\_cpu = px.bar(top\_cpu, x='name', y='cpu\_percent', title='Top 10 Processes by CPU Usage')

                    fig\_cpu.update\_layout(xaxis\_title='Process Name', yaxis\_title='CPU Usage (%)')

                    fig\_cpu.write\_html(os.path.join(self.results\_dir, 'top\_cpu\_processes.html'))

                    logging.info("Memory visualization: Top CPU Processes created.")

                else:

                    logging.warning("No CPU usage data available for visualization.")

                df\_mem = df\_cpu.dropna(subset=['memory\_percent'])

                top\_mem = df\_mem.sort\_values(by='memory\_percent', ascending=False).head(10)

                if not top\_mem.empty:

                    fig\_mem = px.bar(top\_mem, x='name', y='memory\_percent', title='Top 10 Processes by Memory Usage')

                    fig\_mem.update\_layout(xaxis\_title='Process Name', yaxis\_title='Memory Usage (%)')

                    fig\_mem.write\_html(os.path.join(self.results\_dir, 'top\_memory\_processes.html'))

                    logging.info("Memory visualization: Top Memory Processes created.")

                else:

                    logging.warning("No memory usage data available for visualization.")

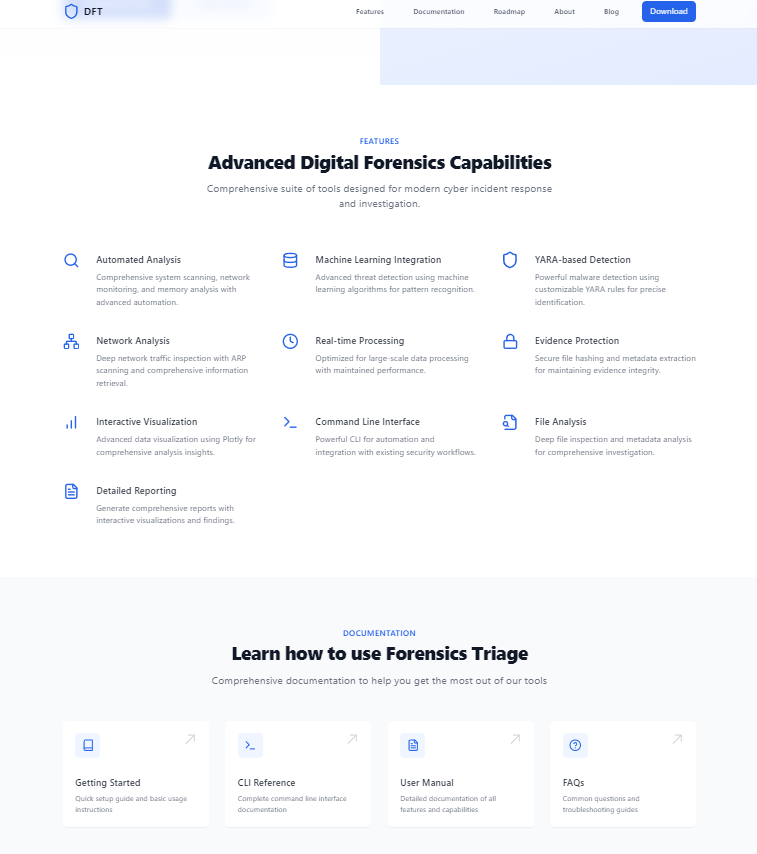
            else:

                logging.warning("No process data available for visualization.")

        except Exception as e:

            logging.error(f"Error creating memory visualizations: {e}")

**Appendix B: Website Preview**

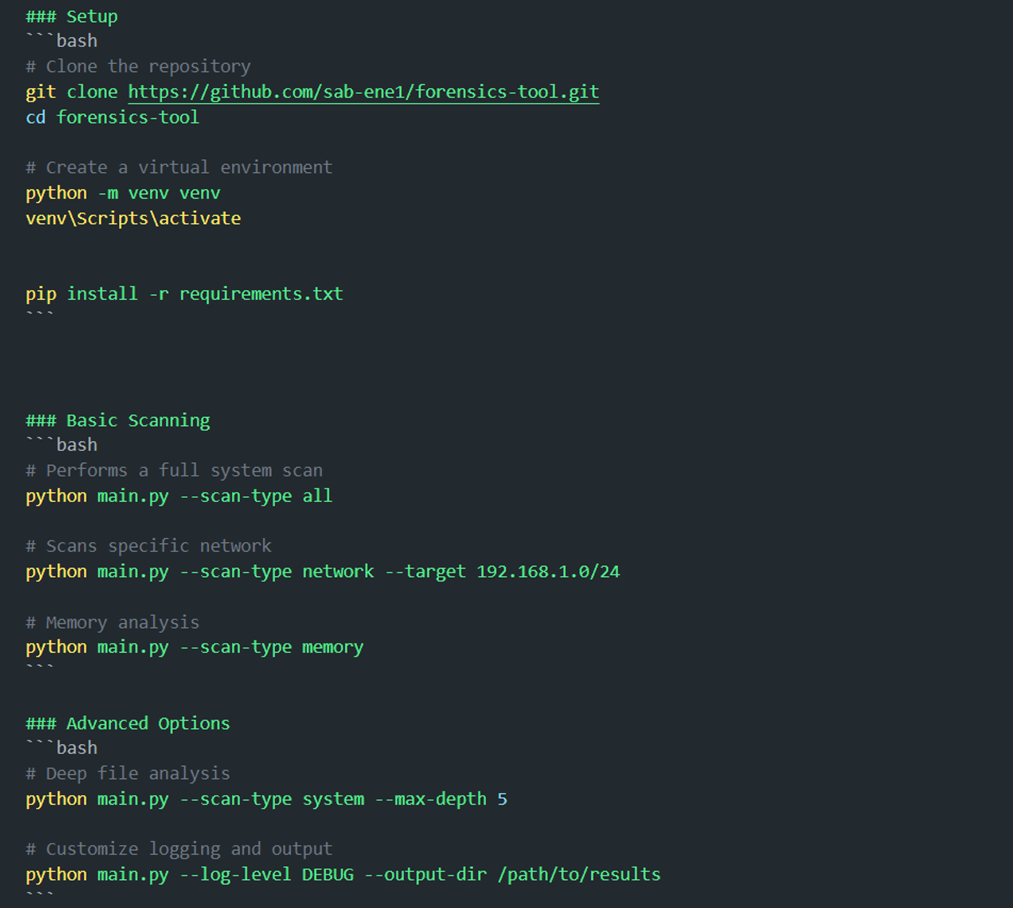
****

**Appendix C: Configuration Instructions**

**1. Prerequisites**

* Install Python 3.8+ and dependencies:

**2. Running the Scanner**

****

**3. Saving Reports**

* exports as HTML format using plotlib python library.