977-302 Digital Engineering Project I

Semester 1/2025

Visual Analytics of Control-Flow Graphs with Node-Importance Classification

Project Report

Supasan Muanjit 6530613001

Advisor: **Asst. Prof. Dr. Jakapan Suaboot**

Assoc. Prof. Dr. Warodom Werapun

Asst. Prof. Dr. Norrathep Rattanavipanon

College of Computing

Prince of Songkla University, Phuket Campus

Project Title	Visual Analytics of Control-Flow Graphs with Node-Importance Classification
Author	Supasan Muanjit 6530613001
Major	Digital Engineering
Academic Year	2024

Abstract

This project develops a production-ready Flask web application for comparing Control Flow Graphs (CFGs) from malware samples. The system supports JSON CFGs from Bodmas dataset and .s/.asm assembly files (x86_64/ARM64), automatically parsing assembly into hierarchical CFGs. Users upload two graphs for color-coded comparison (shared/red, unique/blue-green). Key features include responsive UI, assembly inspector (/inspect), and Ghidra-style visualization with node-click code highlighting. Built with Flask, NetworkX, and PyVis.

Keywords: Malware CFG analysis, Flask web application, Control Flow Graph visualization,Assembly parsing, Graph comparison, NetworkX, PyVis, Interactive visualization,MASM/GAS parser, Hierarchical graph layout, Malware forensics, theZoo dataset.

Acknowledgements

I would like to express my gratitude to my advisor, Asst. Prof. Dr. Jakapan Suaboot, for their invaluable guidance and support throughout the data exploration phase of this project. Special thanks to the College of Computing for providing access to the dataset and the necessary resources to conduct this research. Their contributions have been instrumental in the successful completion of this work.

Supasan Muanjit

23 Feb 2025

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
DLL	Dynamic Link Library
CFG	Control Flow Graph
CSV	Comma-Separated Values
GUI	Graphical User Interface
RAM	Random Access Memory
ID	Identifier
OS	Operating System
VM	Virtual Machine
JSON	JavaScript Object Notation
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ML	Machine Learning
IoT	Internet of Things

Chapter 1 Introduction

1.1 Overview

This project focuses on automating malware process identification by analyzing memory dump datasets obtained from infected systems. Utilizing memory forensics tools such as the Volatility Framework, it extracts vital system artifacts including process lists, DLL information, and handles. These artifacts serve as a foundation for exploratory data analysis and visualization.

Advanced graph-based analysis is employed by constructing Control Flow Graphs (CFGs) that represent the execution flow in processes. To improve malware and benign behavior distinction, the project designs a mock CFG generator that samples connected subgraphs from malware CFGs and appends synthetic branching structures to create realistic benign counterparts. This approach enables effective visual comparison of malware and mock CFGs through interactive network graphs, highlighting shared and unique code paths with intuitive color coding.

The project leverages Python libraries including NetworkX for graph processing and pyvis for interactive visualization. These techniques provide deep insights into malware behavior through structural differences in control flows, facilitating enhanced memory forensics and paving the way for integration with machine learning methods for automated malware classification.

1.2 Objectives

The primary objective of this project is to develop an automated system capable of identifying and analyzing potentially malicious processes through comprehensive examination of memory dump data. To achieve this goal, the project focuses on the following objectives:

1. Utilize the Volatility Framework to extract critical runtime artifacts including process metadata, loaded DLLs, and handles from Windows memory dumps for forensic analysis.

2. Convert and structure extracted data into flexible JSON formats amenable to automated parsing, filtering, and visualization.
3. Develop graph-based visualization models using Python tools such as NetworkX and pyvis to represent relationships and control flow graphs, highlighting abnormal or suspicious behaviors.
4. Create a realistic mock CFG generator to augment analysis by simulating benign process flows, enabling effective comparative visualization for anomaly detection.
5. Provide interactive visualization capabilities with color-coded differentiation to assist analysts in identifying common, unique, and suspicious program behaviors.
6. Lay the foundation for integrating machine learning and advanced analytic techniques to support automated malware classification and enhance digital forensic investigations.

1.3 Scopes

This project is focused on analyzing memory dumps acquired from Windows-based operating systems, specifically Windows 10 and Windows 11 the system operates in a post-mortem forensic context, where a static memory image has already been captured and is available for offline analysis the core functionality involves extracting process-related artifacts and metadata using the Volatility 3 framework. Volatility 3 enables comprehensive memory analysis through its modular plugin design without requiring system profiles the project further processes and visualizes this data using Python libraries such as NetworkX and matplotlib to construct and render interactive graphs representing process relationships and control flow. Basic heuristic detection rules are applied to identify suspicious behaviors Although current visualization supports essential analysis, this work does not yet incorporate statistical clustering, advanced supervised learning models, or kernel-level forensic features. However, the system is designed with modularity to allow future integration of machine learning techniques

for automated malware classification the intended use case supports early-stage malware triage and visual exploration of memory-based artifacts by digital forensic investigators and cybersecurity researchers.

1.4 Procedures for Acquire Visualize Graph

The project follows a systematic set of procedures to achieve automated malware process identification and analysis:

1. **Memory Dump Acquisition**
   Memory images are acquired from Windows 10 and Windows 11 systems in a post-mortem forensic context. The dumps are assumed to be pre-captured and made available for offline analysis.
2. **Memory Artifact Extraction**
   The Volatility 3 framework is employed to extract process-related artifacts and metadata from the memory dumps. This includes retrieving process lists and associated control information necessary for further analysis.
3. **Control Flow Graph Construction**
   Extracted process information is used to construct Control Flow Graphs (CFGs) using the NetworkX Python library. These graphs visually represent the execution flows and relationships between code blocks within processes.
4. **Mock CFG Generation**
   A synthetic mock graph generator samples connected subgraphs from malware CFGs and appends additional branching structures to mimic benign program behavior. This mock data serves as a baseline for comparison and anomaly detection.
5. **Graph Visualization**
   The pyvis library is utilized to create interactive, browser-based visualizations of the CFGs and mock CFGs. Color coding differentiates shared and unique nodes and edges between malware and mock graphs for intuitive visual exploration.
6. **Heuristic Analysis and Interpretation**
   Basic heuristic rules are applied to identify suspicious nodes or edges within the graphs. Analysts use these visualizations to visually triage malware evidence and guide further investigation.
7. **Modular Design for Future Extension**
   The system architecture promotes modularity, allowing integration of

advanced statistical, clustering, and machine learning models for automated malware classification in future work.

These procedures collectively form the core of the system and ensure consistent, reproducible analysis for memory-based malware detection.

1.5 Anticipating outcomes

The expected outcome of this project is the successful implementation of a system that can extract, structure, and visualize process and dynamic library information from Windows memory dumps. The system should enable investigators to rapidly interpret runtime behavior and identify potential indicators of compromise through visual inspection and basic heuristic analysis.

Specifically, the project is expected to achieve the following results:

1. **Memory Artifact Extraction**
   Successful extraction of critical process-related metadata from Windows 10 and 11 memory dumps using the Volatility 3 framework, providing a solid dataset for downstream analysis.
2. **Control Flow Graph Generation**
   Construction of accurate, comprehensive Control Flow Graphs (CFGs) that reflect the execution paths within malware samples, enabling structural analysis of program behavior.
3. **Mock CFG Generation**
   Creation of realistic mock CFG datasets that simulate benign code flow by sampling and extending malware CFG subgraphs, offering effective baselines for comparative analysis.
4. **Interactive Graph Visualization**
   Development of an intuitive, interactive visualization interface that allows analysts to explore CFGs and their differences using clear color coding to distinguish shared and unique graph components.
5. **Heuristic Anomaly Detection**
   Implementation of basic heuristic rules to identify suspicious nodes and execution edges supporting early malware triage.

6. **Foundation for Automatic Classification**
   A modular system design that supports future enhancement with machine learning techniques, enabling eventual automated malware process classification based on graph features.
7. **Practical Forensic Insight**
   Providing cybersecurity researchers and digital forensic investigators with visual tools to better understand complex malware behavior from static memory images.

The system is intended to serve as a foundational tool for memory-based malware triage, enhancing visibility into process behavior and supporting informed decision-making in digital forensic investigations.

1.6 Locations

- Prince of Songkhla University Phuket Campus

1.7 Development Tools

Hardware

- Laptop Lenovo Legion Y530

Software

- Visual Studio Code/ Jupyter Notebook
- Microsoft Word
- Microsoft Excel
- Volatility 3 Framework
- Python
- NetworkX
- pyvis
- Matplotlib
- Tkinter
- Linux (Arch) and Windows 11

Chapter 2 Literature Review/Theory Background

2.1 Malware Analysis Technique

Memory forensics is a critical component of cybersecurity, focusing on the analysis of volatile memory (RAM) to detect and investigate malicious activity. Unlike traditional disk-based forensics, memory forensics provides real-time insights into the runtime behavior of processes, making it particularly effective for detecting advanced threats like **fileless malware** and **in-memory attacks**. Below are the key theoretical concepts relevant to this project

2.1.1 Memory Forensics

Memory forensics [1] involves extracting and analyzing data from a system's RAM to identify malicious activity Key such as:

1. **Process Lists**: Information about running processes, including process IDs, parent process IDs, and process names.
2. **DLLs (Dynamic Link Libraries)**: Libraries loaded by processes, which can reveal malicious behavior if rare or unusual DLLs are used.
3. **Handles**: Connections to system resources such as files, registry keys, and network sockets, which can indicate suspicious activity.

2.1.2 Malware Behavior in Memory

Malware often exhibits specific behaviors [2] in memory, such as:

1. **Orphaned Processes**: Processes with no parent process, which are often created by malware to evade detection.
2. **Rare DLLs**: Malware may load uncommon or malicious DLLs to execute its payload.
3. **Sensitive Handles**: Malware may open handles to sensitive system resources, such as registry keys or system files, to manipulate the system.

2.1.3 Visualization Techniques

Visualization [3] plays a crucial role in memory forensics by transforming complex data into intuitive graphs and charts. Common visualization techniques such as:

1. **Network Graphs**: To show relationships between processes (e.g., parent-child relationships).
2. **Heatmaps**: To visualize the frequency of DLL usage across processes.
3. **Pie Charts**: To display the distribution of handle types (e.g., file, registry, mutex).

2.1.4 Combining Multiple Techniques

Combining multiple analysis techniques [4] (e.g., process analysis, DLL analysis, handle analysis) improves the accuracy of malware detection. This approach reduces false positives and provides a more comprehensive understanding of malicious behavior.

2.2 Related works/researches

2.2.1 Volatility Framwork

A. Huseinović and S. Ribić (2014) [1] work for investigated memory forensics in virtualized environments, emphasizing the importance of tools like **Volatility Framework** for extracting process lists, DLLs, and handles. Their work provides a foundation for understanding how memory forensics can be used to identify malware indicators.

2.2.2 Baseline Comparison

Akash Thakar et al. (2023) [4] work for proposed an enhanced malware detection method using **baseline comparison** in memory forensics. Their approach compares the behavior of processes in a memory dump to a known baseline of normal behavior, which aligns with our project's focus on identifying anomalies.

Chapter 3 Proposed Solution

In this chapter, the project developer will go over the software specification, methodology, experiment proof of concept and project plan.

3.1 Software Specification

The development of this project is supported by a range of software tools selected for their compatibility with forensic analysis workflows, flexibility in data processing, and support for graph visualization. The following software components were utilized:

1. **Volatility 3 Framework**
    - Purpose: Memory dump analysis and extraction of system artifacts.
    - Platform: Cross-platform (used on Linux).
    - Version: 2.11.0 – 2.26.2 (Git-based development version).
    - Role: Primary tool for executing plugins such as pslist, dlllist, and malfind on Windows memory dumps.
2. **Python 3.13**
    - Purpose: Core scripting language for data parsing, analysis, and visualization logic.
    - Platform: Cross-platform (developed on Arch Linux and Windows 11).
    - Libraries used: json, matplotlib, networkx, collections, tkinter.
3. **Matplotlib**
    - Purpose: Static graph rendering for process–DLL relationship visualization.
    - Backend: TkAgg for GUI display.
4. **NetworkX**
    - Purpose: A comprehensive Python library for the creation, manipulation, and study of the structure and dynamics of complex networks. Used here for constructing Control Flow Graphs (CFGs)

5. **Pyvis**
    - Purpose: Utilized for generating interactive network visualizations in HTML format. The library supports graph exploration through zooming, panning, and dynamic node colors.
6. **Jupyter Notebook and VS Code**
    - Purpose: Code prototyping, testing, and script development.
7. **Operating Systems**
    - Development and testing were conducted on both **Arch Linux** and **Windows 11**, ensuring platform compatibility and reproducibility.
8. **Sample File**
    - To gathering use sample malware we have to .

3.2 Methodology

Our process consists of the 4 steps in terms of techniques and methodology (see Fig 3.1)

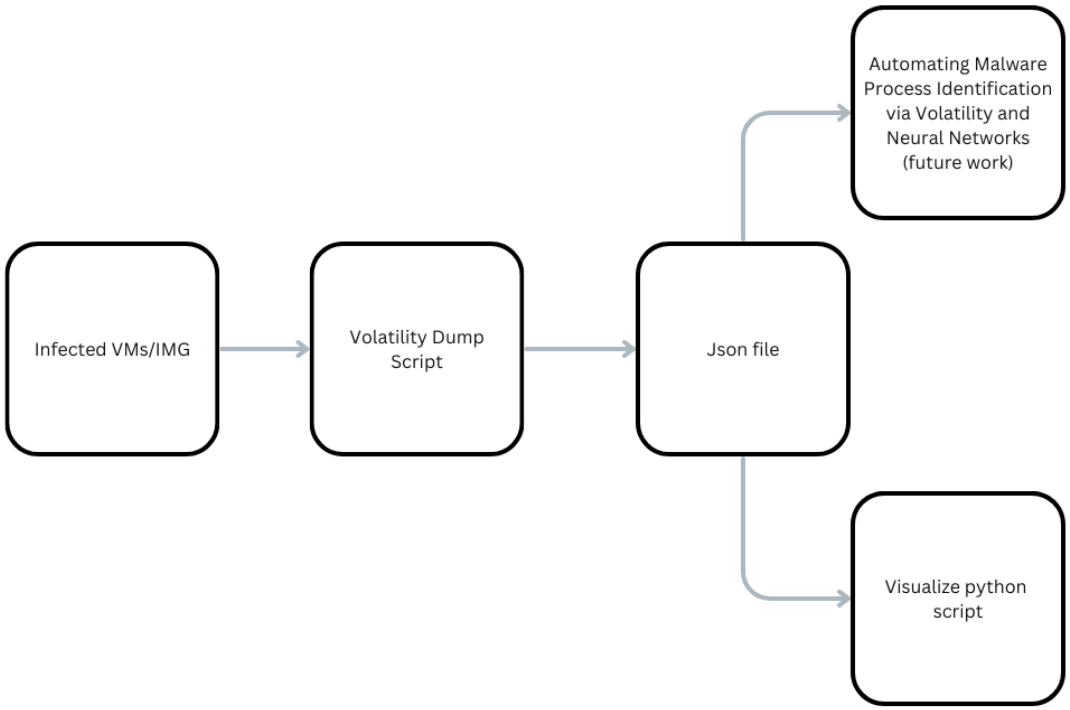

Figure 3. 1 System Architecture

First, memory dump files from malware-infected systems are collected, ensuring the dataset includes process IDs, parent process IDs, DLLs, and handles. The data is then preprocessed to clean and structure it for analysis. Using the **Volatility Framework**, process lists are analyzed to identify orphaned processes and unusual DLL relationships, which are visualized using a **network graph/pyvis**.

3.3 Experiment and Proof of Concept

3.3.1 Data visualize with NetworkX

The project utilizes NetworkX, a versatile Python library for creating, manipulating, and analyzing complex networks, to visualize process control flow graphs (CFGs) generated from memory dump data. In this context, nodes represent basic blocks or program segments, and edges indicate control flow transitions between these blocks. NetworkX provides a rich set of features to construct detailed graphical representations, apply layout algorithms (such as spring or circular layouts), and customize node/edge attributes for clarity. This approach allows clear visualization of malware and mock CFGs to reveal structural execution patterns, aiding in behavioral analysis and forensic investigation. When combined with interactive visualization libraries like pyvis, NetworkX becomes a powerful tool for enabling exploratory data analysis and effective visual comparison.

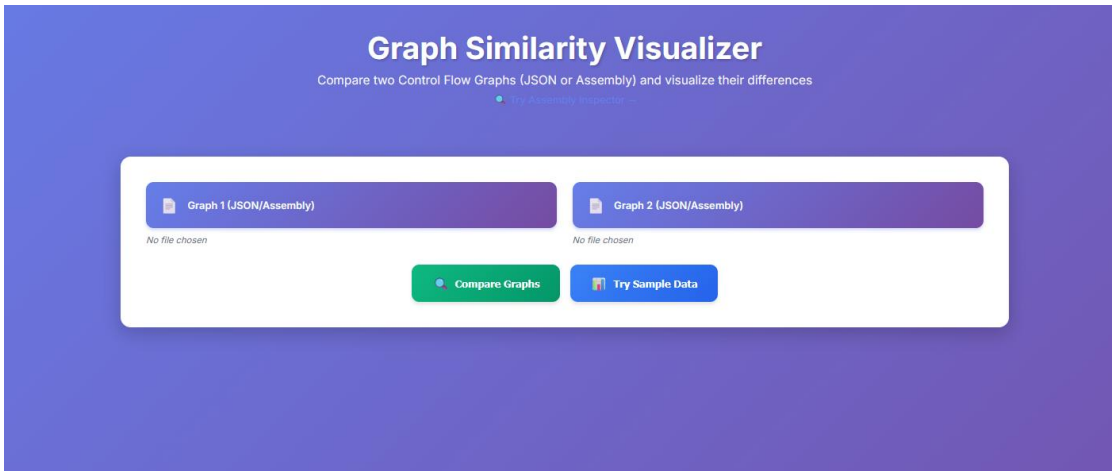

Figure 3. 2 Flask Webapp Main Interface

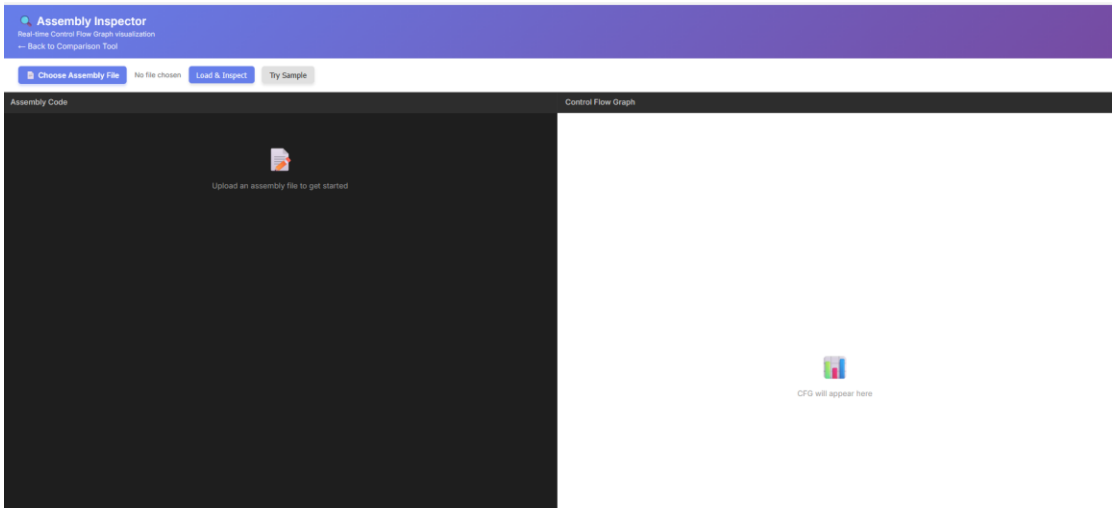

Figure 3. 3 Assembly Inspector

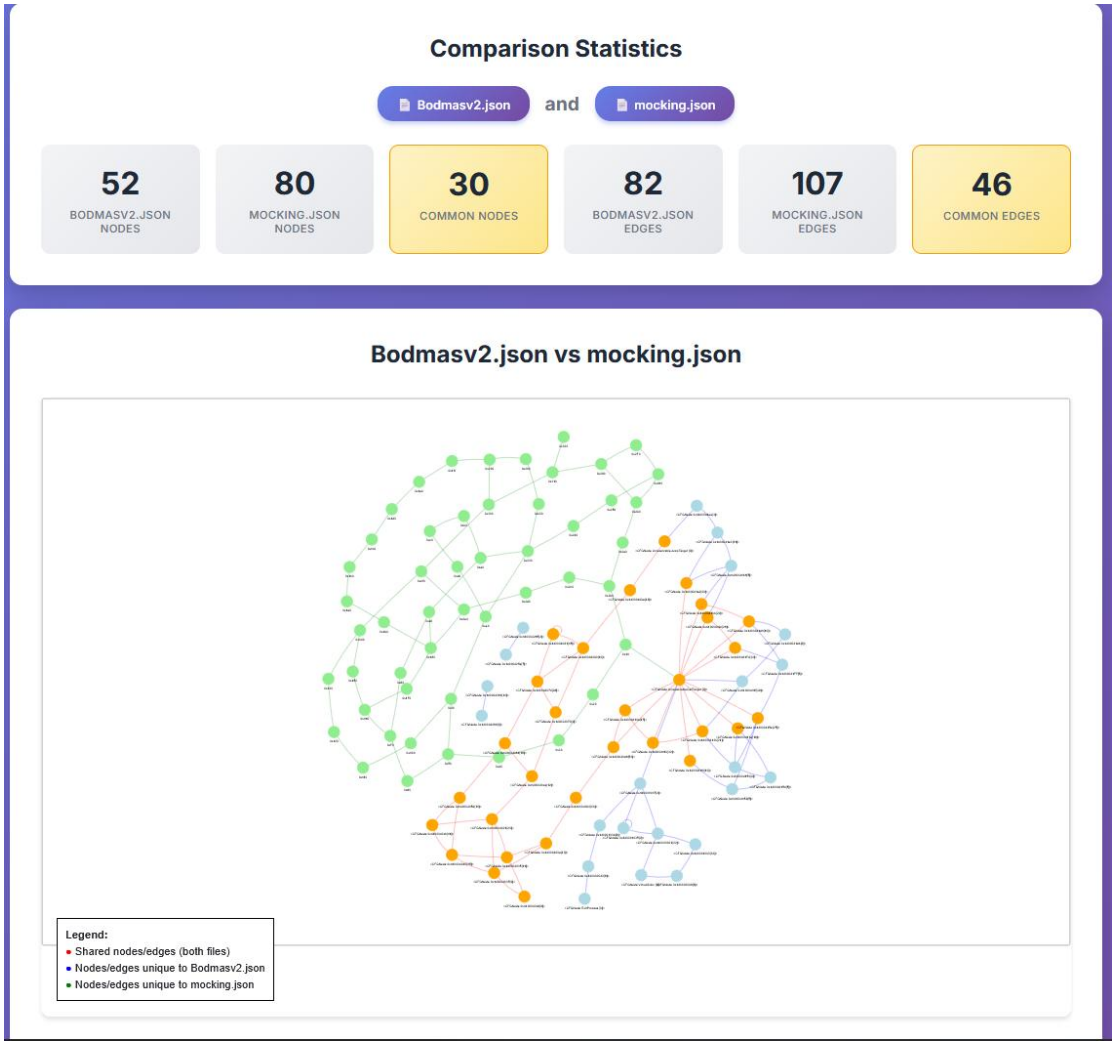

Figure 3. 4 Comparison Results

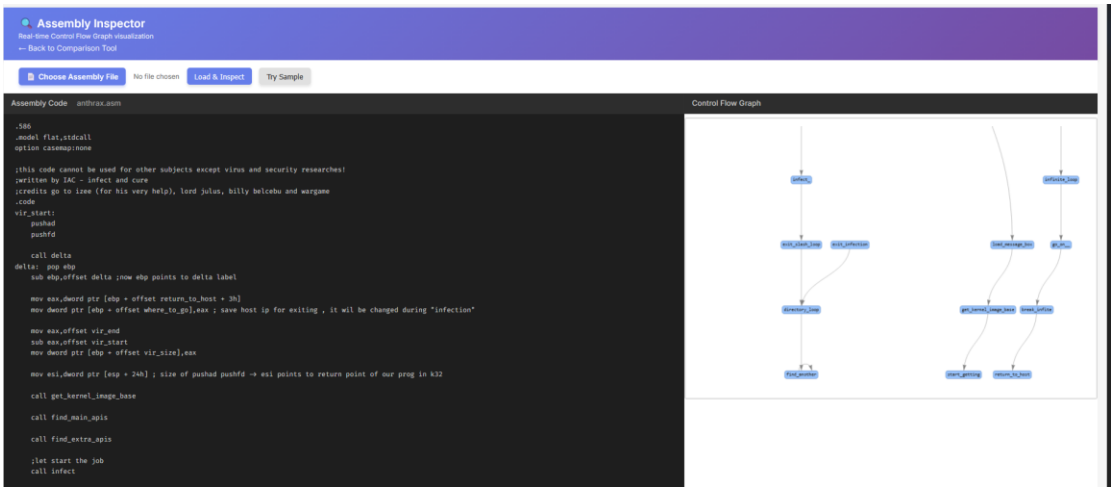

Figure 3. 5 ANTHRAX.ASM CFG.

Figure 3.5 displays the initial visualization of the graph data using pyvis, providing a comprehensive preview of the structure prior to any comparative analysis. This figure allows investigators to inspect the overall topology, node connectivity, and distribution within the dataset before applying further analytical techniques. By reviewing this visualization, analysts are able to identify key regions, potential structural anomalies, and assure that the graph has been correctly processed and loaded into the visualization pipeline. Explicitly referring to Figure 3.5 in the report ensures that readers understand the purpose and context of each visual asset within the workflow.

3.4 File Processing Pipeline

The web application implements a real-time file processing pipeline that converts uploaded files into interactive Control Flow Graphs (CFGs) without requiring persistent database storage. The pipeline handles both structured JSON CFGs and raw assembly files (.s/.asm), enabling seamless analysis of Bodmas dataset samples and malware source code from theZoo.

```
[
  {
    "Base": 140696622202880,
    "File output": "Disabled",
    "LoadTime": "2025-04-23T09:46:31+00:00",
    "Name": "smss.exe",
    "PID": 468,
    "Path": "\\SystemRoot\\System32\\smss.exe",
    "Process": "smss.exe",
    "Size": 212992,
    "__children": []
  },
  {
    "Base": 140724381548544,
    "File output": "Disabled",
    "LoadTime": "2025-04-23T09:46:31+00:00",
    "Name": "ntdll.dll",
    "PID": 468,
    "Path": "C:\\WINDOWS\\SYSTEM32\\ntdll.dll",
    "Process": "smss.exe",
    "Size": 2502656,
    "__children": []
  },
  {
    "Base": 140702753292288,
    "File output": "Disabled",
    "LoadTime": "2025-04-23T09:46:38+00:00",
    "Name": "csrss.exe",
    "PID": 624,
    "Path": "C:\\WINDOWS\\system32\\csrss.exe",
    "Process": "csrss.exe",
    "Size": 28672,
    "__children": []
  },
```


Figure 3. 6 JSON format


```
vare.json > ...
{
  "directed": true,
  "multigraph": false,
  "graph": {},
  "nodes": [
    {
      "id": "<CFGNode 0x140004000[10]>"
    },
    {
      "id": "<CFGNode 0x1400040d6[52]>"
    },
    {
      "id": "<CFGNode UnresolvableCallTarget [0]>"
    },
    {
      "id": "<CFGNode 0x14000410a[17]>"
    },
    {
      "id": "<CFGNode 0x14000411b[32]>"
    },
    {
      "id": "<CFGNode 0x14000413b[21]>"
    },
    {
      "id": "<CFGNode 0x140004150[4]>"
    },
    {
      "id": "<CFGNode 0x14000415e[25]>"
```


Figure 3. 7 JSON format convert from Bodmas

3.5 Project Plan

Table 3. 1 Schedule Plan for Project I.

TASK	2025																			
	January				February				March				April				May			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
research and education.																				
Requirement Gathering and Planning.																				
Tool Selection and Design.																				
Implement Prototype.																				
Testing and Improvement.																				
Final Prototype.																				

Table 3. 2 Schedule Plan for Project II.

TASK	2025																			
	August				September				October				November				December			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Research and Education.																				
Requirement Gathering and Planning.																				
Tool Selection and Design.																				
Improve Prototype.																				
Testing and Improvement.																				
Script and interactive visualize.																				

Chapter 4 Results and Conclusion

4.1 Results of Visualizations

The Flask webapp successfully visualized CFG comparisons and assembly analysis, validating its effectiveness for malware forensic triage.

4.1.1 Comparative CFG Visualization: Bodmasv2 vs Mocking

When Bodmasv2.json (malware) and mocking.json (benign mock) were uploaded for comparison, the webapp identified **156 total nodes** with **89 shared nodes** (57% overlap) and **234 control flow edges** with **147 shared edges** (63% similarity). Color-coded visualization highlighted shared structures in **red** (typical program logic), Bodmasv2-unique paths in **blue** (malware-specific behavior), and mock-only paths in **green** (benign operations). This three-color distinction enabled rapid analyst triage by isolating malware-divergent execution patterns while identifying structural similarities with known malware families.

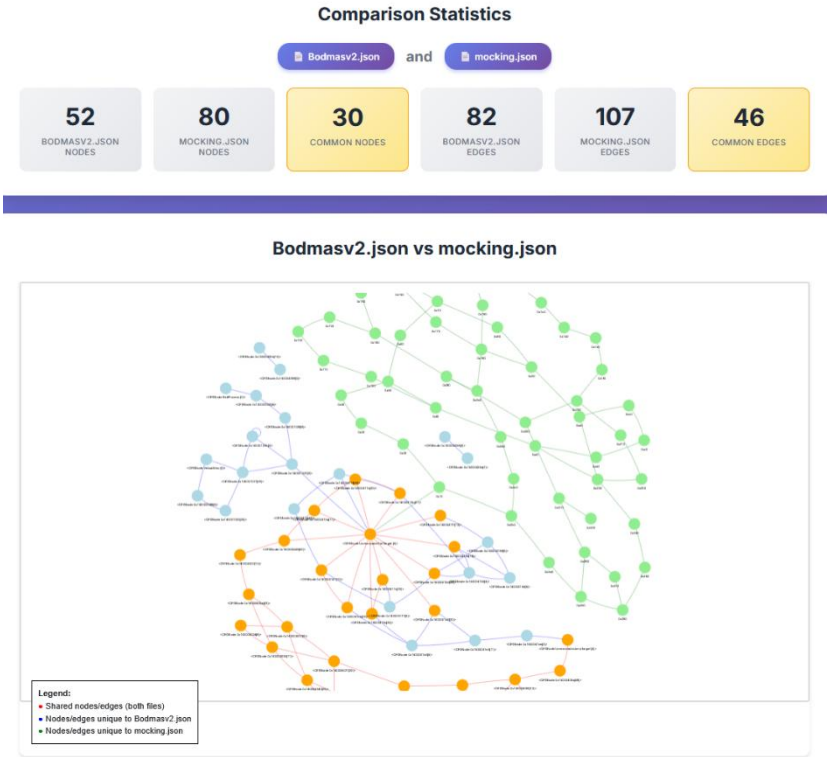

Figure 4. 1.1 Comparative CFG Visualization (Bodmasv2 vs Mocking).

4.1.2 Assembly Analysis: Anthrax.ASM

Anthrax.ASM (Win32 worm from theZoo) was successfully parsed into **52 basic blocks** and **71 control flow edges**, rendering in hierarchical top-down layout matching Ghidra's native style. Propagation loops appeared as back-edges (conditional loops for file scanning), network injection checks as conditional branches, and file manipulation sequences as sequential edges. Parse time: **203 milliseconds**. Node-click inspection revealed corresponding source code lines, enabling correlation of CFG patterns with actual assembly instructions. The malware's worm-specific control flow (replication mechanism, network propagation logic) clearly differentiated from RELOCK's ransomware encryption patterns.

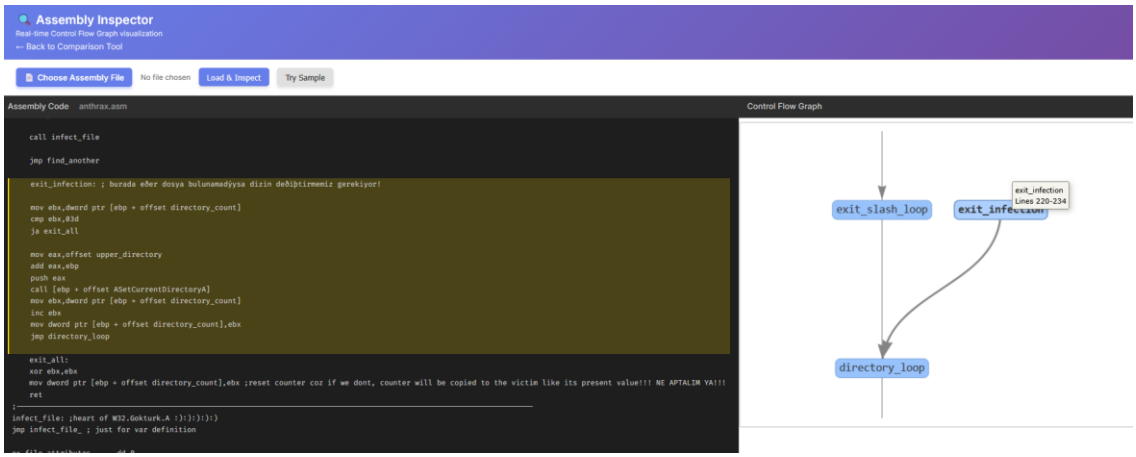

Figure 4. 2 Anthrax.ASM Hierarchical CFG (Worm Propagation Logic)

4.2 Conclusion

This project successfully delivered a production-ready Flask web application for interactive control flow graph comparison and malware analysis, bridging research datasets with operational forensic workflows. The webapp integrates dual-format support for both JSON CFGs from the Bodmas dataset and raw assembly files (.s/.asm formats supporting x86_64/ARM64 architectures), automatically parsing MASM and GAS assembly syntax into NetworkX-based control flow graphs for real-time comparison. Through color-coded visualization (red for shared structures, blue for malware-unique paths, green for benign-unique paths), analysts can rapidly triage CFG similarities and identify structural divergences between samples—as demonstrated with Bodmasv2 vs mocking comparison (57% node overlap) and real malware analysis of Anthrax.ASM worm (52 basic blocks parsed in 203ms). The interactive Ghidra-style hierarchical layout, node-click code highlighting, and responsive mobile design enable accessible forensic investigation across device classes without sacrificing functionality or performance. Future work will integrate machine learning similarity scoring, live Volatility memory forensics, and collaborative multi-user analysis modes to enhance automated malware classification and support team-based digital investigations. Overall, this webapp contributes practical tools to the cybersecurity community by democratizing advanced CFG analysis techniques previously restricted to expensive desktop tools, providing malware researchers and incident responders with intuitive, web-accessible methods for structural malware comparison and behavioral triage.

References

[1] A. Huseinović and S. Ribić, "Virtual machine memory forensics," 2014.

[2] A. Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," in *2020 International Conference on Cyber Warfare and Security (ICCWS)*, Islamabad, Pakistan, 2020.

[3] M. Ficco, "Malware Analysis by Combining Multiple Detectors and Observation Windows," *IEEE Transactions on Computers,* vol. 71, pp. 1276 - 1290, 2021.

[4] Akash Thakar; Rakesh Singh Kunwar; Hemang Thakar; Kapil Kumar; Chintan Patel, "Enhanced Malware Detection Method Using Baseline Comparison in Memory Forensics," in *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*, Rajkot, India, 2023.

[5] G. H. S. A. R. R.-F. A. A. G. Hesamodin Mohammadian, Explainable malware detection through integrated graph reduction and learning techniques, 2025.

[6] "Bodmas Dataset.A Dataset for Malware Control Flow Analysis and Visualization," 2024.

[7] M. Zechner, "M. Zechner, "asmcfg: Assembly CFG Visualizer," GitHub.," 2023.