

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

Supasan Muanjit

Advisor Asst. Prof. Dr. Jakapan Suaboot

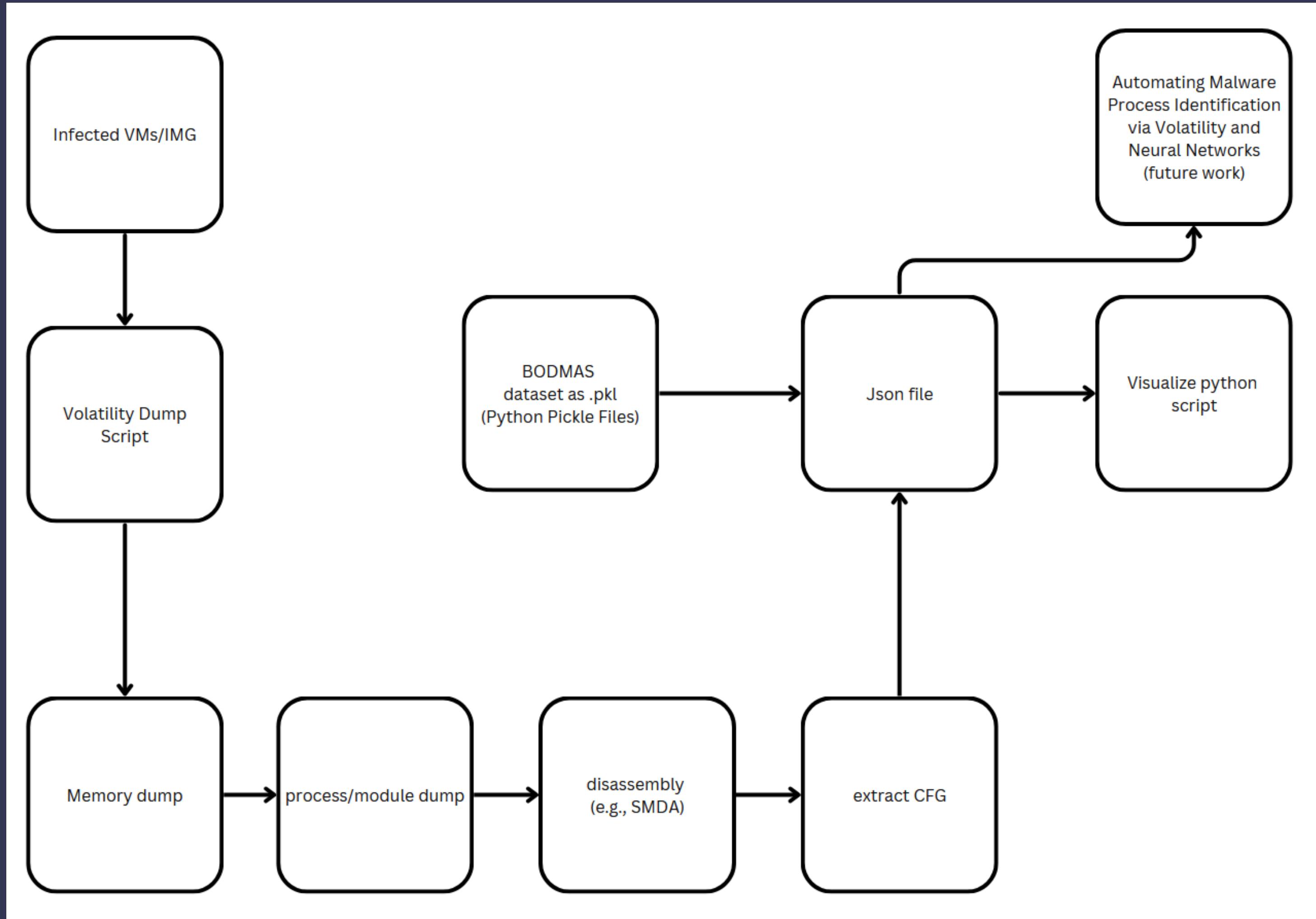
Project Overview

- Analysis and visualization of malware behavior from pre process data
- Use of Python libraries (pandas, matplotlib, NetworkX, pyvis)
- Objective: Identify malicious process indicators visually and structurally

Objectives

- Extract process metadata from memory dumps using Volatility Framework
- Construct Control Flow Graphs (CFGs)
- Provide interactive visualization for malware detection
- Lay foundation for machine learning integration

Methodology



- **Memory Acquisition** Infected VMs or images are processed with Volatility to obtain raw memory dumps.
- **Process/Module Extraction** From the memory dumps, individual processes or modules suspected of being malware are isolated and dumped.
- **Disassembly** Tools such as SMDA (or similar) are used on the module dumps to disassemble executable code, allowing for static analysis.
- **CFG Extraction** Disassembled code is analyzed to extract the Control Flow Graph (CFG), which represents the execution paths within the malware's code.
- **Data Transformation** The resulting data—including CFGs—is structured into JSON files, enabling easier downstream processing and visualization.
- **Dataset Integration** Alternatively, pre-existing datasets such as BODMAS can be loaded in Python pickle (.pkl) format and converted into the same JSON structure for consistency.
- **Visualization and Automation** The JSON data can be visualized using Python scripts. The final goal is to automate malware process identification using machine learning techniques, with Volatility and neural networks suggested as future work.

Data Exploration

Dataset Overview

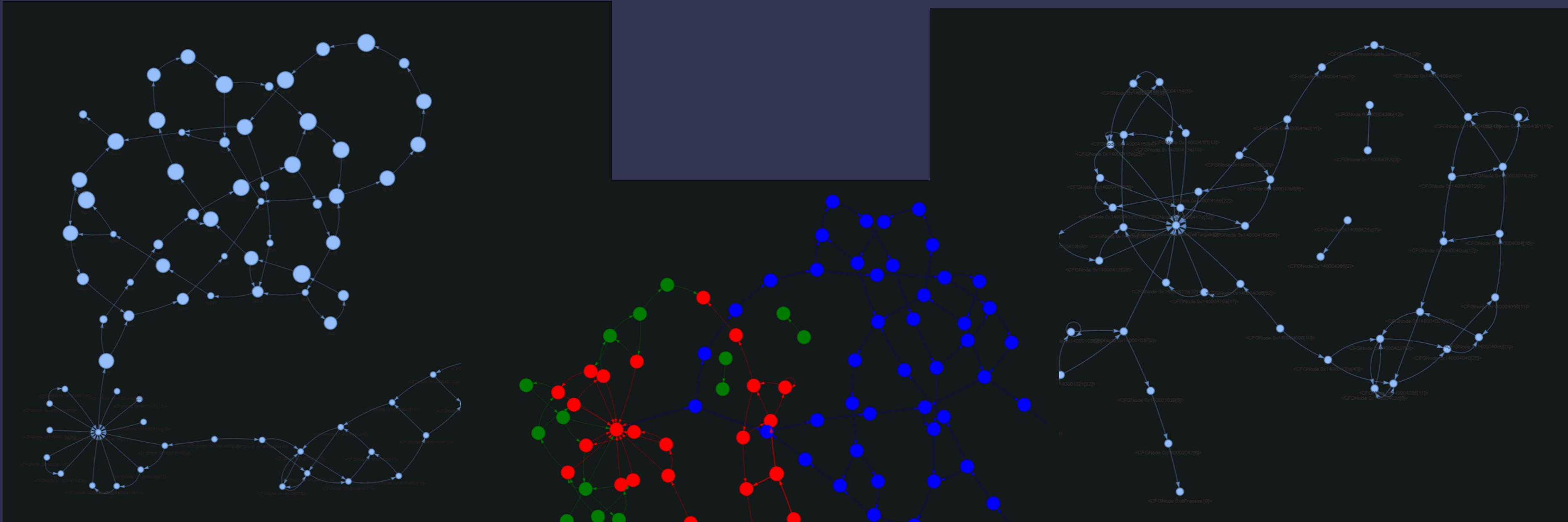
Name	Last modified	Size	Description
Parent Directory			
0a3b4a3ae8a088877a28a44e14d2a235618784f482e45b268dda47d1f34168ea.pkl	2024-10-28 18:51	256M	
0ac72e64ab8a92080428392a1297eb71b84d59a067a10e16d8fb5a93cb815979.pkl	2024-10-28 18:51	251M	
0b29e229a56fc43b6022f3cd87008ded7360a5f3e4c98c61dd9c4222a90de24.pkl	2024-10-28 18:51	250M	
0b56c37bd80c5242fca24fa9b201606824f214d8ada7fa6442eae6a6c654c779.pkl	2024-10-28 18:51	16M	
0bfed924fd2493ef3c481a8eb2d2e4a4b11811705fb190136244c1cfac5db09.pkl	2024-10-28 18:51	253M	
0c6d6bf8036d59d2bc3882143eb86c7b0bcc3d87eee536e0c782f414a19ad712.pkl	2024-10-28 18:51	274M	
0cda7c8876136568abdb86a38b76898490ced48bfa1ac75889a7eca248155280.pkl	2024-10-28 18:51	226M	
0d7b74dc63413d1ade59fcf663f16fca483605070c546a30ead086b31441f819.pkl	2024-10-28 18:52	199M	
0da09a18dd5a491e54384bd70b7bb635b81112531a0ef3c58c26c2ae8b1704d0.pkl	2024-10-28 18:52	113M	
0ecc8c26e836b3d60a418c404138c661e2b3a49f01aa41a2604d729537a0a68c.pkl	2024-10-28 18:52	286M	

The screenshot shows a code editor window with the file "malware.json" open. The file path is displayed as "C: > Users > Admin > Documents > Work > Project > CFGs_y4t1 > malware.json". The code itself is a JSON object representing a graph structure:

```
1  {
2      "directed": true,
3      "multigraph": false,
4      "graph": {},
5      "nodes": [
6          {
7              "id": "<CFGNode 0x140004000[10]>"
8          },
9          {
10             "id": "<CFGNode 0x1400040d6[52]>"
11         },
12         {
13             "id": "<CFGNode UnresolvableCallTarget [0]>"
14         },
15         {
16             "id": "<CFGNode 0x14000410a[17]>"
17         },
18         {
19             "id": "<CFGNode 0x14000411b[32]>"
20         },
21         {
22             "id": "<CFGNode 0x14000413b[21]>"
23         },
24         {
25             "id": "<CFGNode 0x140004150[4]>"
26         },
27         {
```

Dataset from BODMAS

Data Virtualization



Legend:

- Shared nodes/edges (both files)
- Nodes/edges unique to mocking.json
- Nodes/edges unique to boombasv2.json

*Each node's name identifies a specific execution block or function within the analyzed process

*The naming allows forensic analysts to trace which section of code is being executed, whether it's benign or potentially malicious, supporting targeted investigation and cross-referencing with known malware signatures

Demonstration

Summary

What has been Done

Constructed Control Flow Graphs (CFGs)
representing malware execution paths using
NetworkX.

Created interactive visualizations of CFGs with
pyvis, showing shared and unique code paths
with color coding.

Implemented basic heuristic anomaly detection
to highlight suspicious nodes and edges.

What can improve

Do web version so it can be easy to access

Reference

- [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.

Q&A