University of Guelph

Computer Science Department

CIS*6530 - Threat Intel & Risk Analysis

Extracting Opcodes: Researching APT group set 5

Submission 3

(10%)

Team Members

Nafis Ahmed Awsaf | nawsaf@uoguelph.ca | 1402517

Jacob Drobena | jdrobena@uoguelph.ca | 0969071

Work Breakdown

Nafis | Pages 23-41 (19 APTS)

Jacob | Pages 3-22 (19 APTS)

Professor: Dr. Ali Dehghantanha

Deadline: Oct 30nd, September 2025

Table of Contents:

I. Introduction:

This project re-engineers an existing open-source Ghidra opcode extraction tool originally designed to export instruction opcodes into CSV format [1]. This tool has been refactored and extended to align with the current task and to handle large-scale malware datasets, producing lightweight .opcode files and, when not resource-constraining, an accompanying, more detailed .csv output. The improvements allowed for manual adjustments to resource throttling to improve

storage efficiency and analysis speed. The emphasis on this task was to highlight automation; however, there were constraints related to time delays and output size, where the pulled malware sample would either produce an output too large to add to the GitHub repo or take too long per operation. Hence, automation was conducted in batches to output as many malware samples as possible within an appropriate runtime.

This lab report will cover the following items:

1. Lab Environment and Testbed Configuration will note VM specifications, software stack, network topology, and snapshot details during experimentation.
2. The Method of Procedure (MOP) outlines the steps for the process used to refactor, test, and execute the opcode extraction workflow, including the design changes, optimization efforts, and struggles.
3. Execution Evidence will show logs and screenshots verifying system operation.
4. Analysis and Discussion highlight the results, performance trade-offs, and potential improvements.

II. Lab environment:

The experiment was executed on several cloned Kali Linux 2018.4 virtual machines running in VMware Workstation. Each VM instance was provisioned from the same standardized base image, which included the necessary dependencies, configuration files, and automation scripts located in the repository's [full_kali_run] directory.

In this controlled setup to ensure reproducibility, each virtual machine was configured to run Ghidra 11.4.2 (Public Release) in headless mode, enabling automated opcode extraction without invoking the graphical interface.

VM Specifications:
Base OS: Kali Linux 2018.4 (64-bit)
Hypervisor: VMware Workstation (Pro or Player)
Processor: 4
Memory Allocation: 8–12 GB ram
Network Mode: Host-Only when executing the script, or NAT when we were pulling data from GitHub.
Reverse Engineering Framework: Ghidra 11.4.2 PUBLIC

,

III. Environment Setup:

This section outlines the configuration steps used to install Ghidra and prepare the environment for headless operation on a fresh Kali VM.

1. Installing Java JDK 21 (Required for Ghidra)

Ghidra requires a compatible Java Runtime Environment. The Temurin 21.0.9+10-LTS (x64) version was used for compatibility and stability [2].

Installation steps:
sudo apt update
sudo apt install -y wget unzip
wget https://download.bell-sw.com/java/21+10/bellsoft-jdk21+10-linux-amd64.deb
sudo dpkg -i bellsoft-jdk21+10-linux-amd64.deb

2. Installing Ghidra

Download Ghidra from the official NSA GitHub repository []:
https://github.com/NationalSecurityAgency/ghidra

mkdir -p ~/ghidra
cd ~/ghidra
wget
https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_11.4.2_build/ghidra_11.4.2_PUBLIC_20250826.zip
unzip ghidra_11.4.2_PUBLIC_20250826.zip -d ~/ghidra
cd ~/ghidra/ghidra_11.4.2_PUBLIC/
chmod +x ghidraRun
chmod +x support/analyzeHeadless

3. Additional dependencies:

sudo apt install -y python3-pip
python3 -m pip install tqdm

IV. Method of Procedure:

This section outlines the systematic process for reworking and executing the opcode extraction pipeline, assuming proper installation or pull from the provided full_kali_run.

1. Baseline setup from the original GitHub tool that outputted CSV-only results was previously two files: get_opcode.py and the Jython caller ghidra_opcode_script.py to perform the full output to the CSV opcode extraction tool

2. The refactoring process merged these two into a single Python file [ghidra_opcode_script.py] for simplified execution, maintenance, and extension.
    a. Introduced timeout functionality
    - DEFAULT_TIMEOUT_SECONDS = 150 # for seeing if it can pull CSV
    - OPCODE_ONLY_TIMEOUT_SECONDS = 500 # maximum allotted time before terminating current file test
    b. Failure handling and progress logging introduced ‘check_manually.txt’ to track any binaries that exceeded both full and opcode-only timeouts.

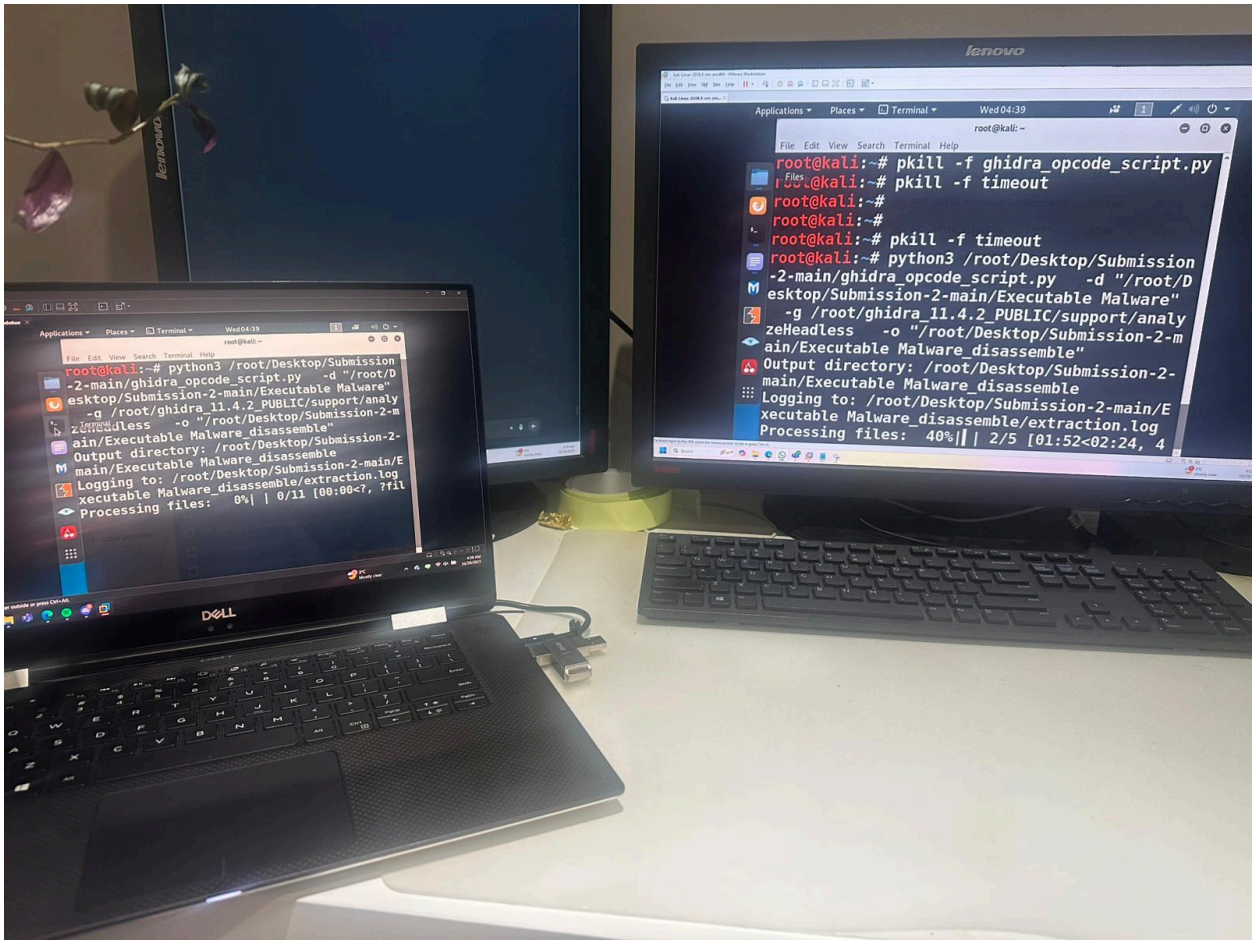
   c. Adjustments to the parallelization from the original github code to reduce code complexity. Default configuration for our project set DEFAULT_WORKERS=1 to avoid VM instability.\

3. Execution Workflow

Once the environment and refactored script were ready (provided in the existing GitHub), the following step-by-step procedure was used to execute the opcode extraction:

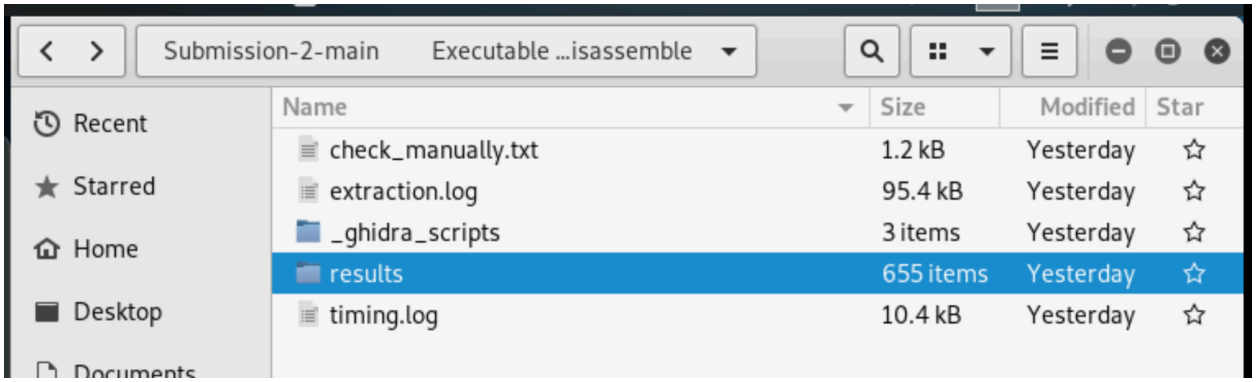
1. Cmd execution:
   root@kali:~# python3 /root/Desktop/Submission-2-main/ghidra_opcode_script.py -d "/root/Desktop/Submission-2-main/Executable Malware" -g /root/ghidra_11.4.2_PUBLIC/support/analyzeHeadless -o "/root/Desktop/Submission-2-main/Executable Malware_disassemble"
2. The script scanned the specified directory for valid binaries (.exe, .dll, .so, .bin, etc.).
3. If the hash was new, the script initialized a temporary Ghidra project and launched the headless analysis.
4. The __ghidra_opcode_script_full_.py or __ghidra_opcode_script_opcode_.py script was dynamically generated and injected into Ghidra's runtime.
5. Results were exported into /results/<prefix>/<filename>.opcode.
6. Upon completion or timeout, temporary project files were automatically cleaned up.

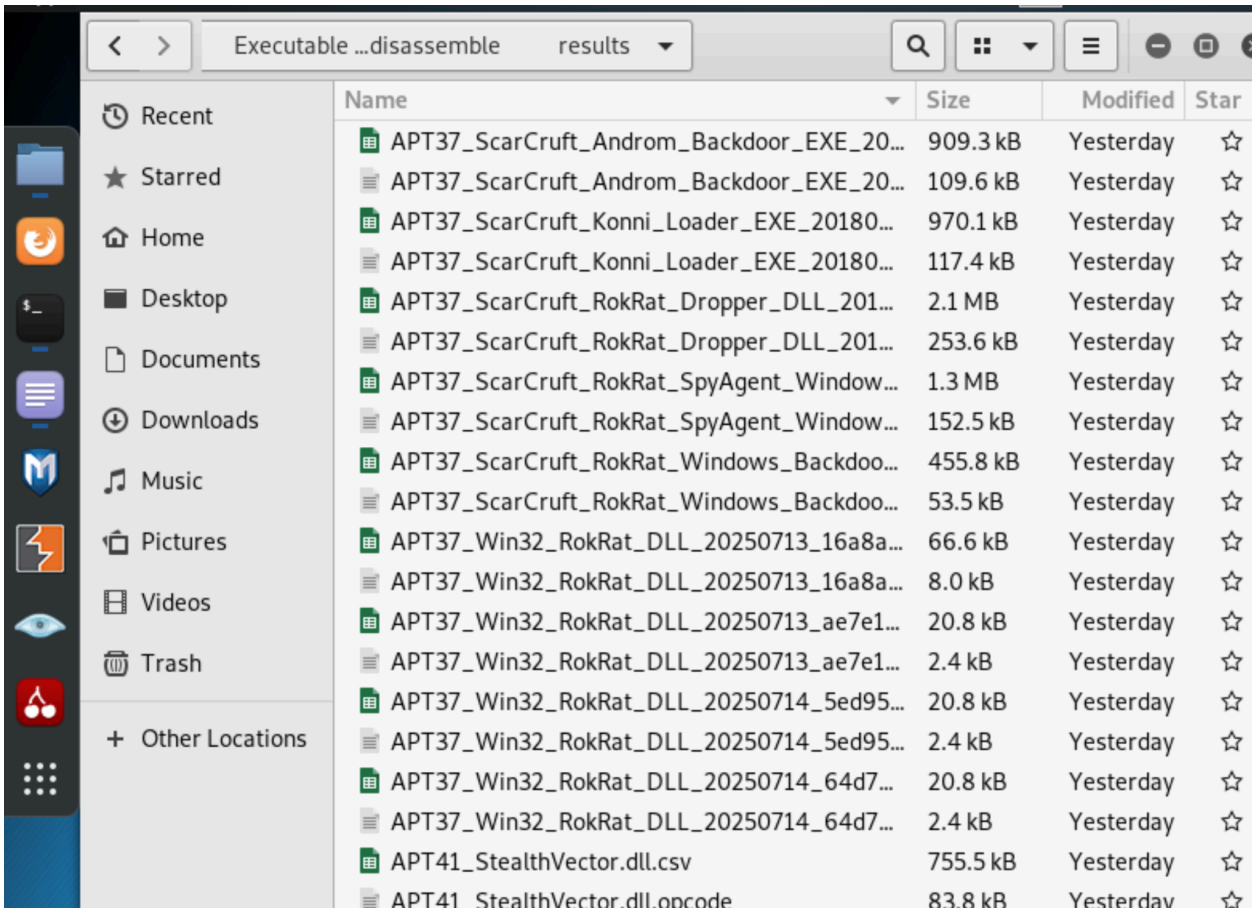
Since deduplication was built in, rerunning the same command would skip already processed or duplicated samples, continuing only from unprocessed ones.



After execution, the system generated the following output files and directories:

Executable Malware_disassemble/
├── extraction.log
├── results/
│ ├── AP/...
│ │ ├── APT_C-36_Windows_AgentTesla_....opcode
│ │ └── ...
│ ├── check_manually.txt
│ ├── processed_hashes.txt
└── _ghidra_scripts/
├── __ghidra_opcode_script_full__.py
└── __ghidra_opcode_script_opcode__.py





VIII. Conclusion:

The method of procedure successfully transformed the original CSV-oriented extraction process into an automated, efficient, and fault-tolerant pipeline capable of handling large malware datasets. The reworked tool enables reproducible, scalable extraction while minimizing system resource usage. Automation through batch execution, integrated hash deduplication, and comprehensive logging ensures that every operation is traceable, resumable, and optimized for long-term research use.References:

[1] bolin8017, Github, “OpCodeReverseTool” May, 2024
https://github.com/bolin8017/OpCodeReverseTool
[2] Java JDK 21
https://adoptium.net/download?link=https%3A%2F%2Fgithub.com%2Fadoptium%2Ftemurin21-binaries%2Freleases%2Fdownload%2Fjdk-21.0.9%252B10%2FOpenJDK21U-jdk_x64_linux_hotspot_21.0.9_10.tar.gz&vendor=Adoptium
[3] Ghidra NSA GitHub https://github.com/NationalSecurityAgency/ghidra