SCHOOL OF
COMPUTER SCIENCE

Course: CIS*6530 (01) F24 - Threat Intel & Risk Analysis

Technical Report on Malware Opcode Extraction Using GHIDRA

Group 5 Members

Bisrat Dereje Kura (1313736)
Oluwaseun Omale (1354621)
Zuhaibuddin Bhutto (1351434)

November 4, 2024

1 Introduction

In this project, I used the reverse-engineering tool **GHIDRA** to extract operational codes (opcodes) from malware samples. Opcodes are the low-level instructions representing the actions performed by a program, making them valuable for understanding malware behavior. By extracting these opcodes, we can create datasets to help train and test AI agents in identifying and analyzing malware.

This report details the steps taken to extract opcodes from malware samples using GHIDRA, including the Python scripts developed for automation. The output, which includes opcode files and scripts, was submitted to our group's GitHub repository as required.

2 Tools and Environment

- **GHIDRA**: A reverse-engineering software developed by the NSA, widely used for analyzing executable code. It allows for scripting with Java or Python for code automation.
- **Python**: Python scripts were used to manage and automate the extraction of opcodes by interfacing with GHIDRA.
- **Operating System**: Some of us used Windows 10, Linux, and MacOs with **Java Development Kit (JDK 21+)** for GHIDRA script compatibility.
- **Controlled Environment**: Malware samples were handled in a secure, offline environment with antivirus measures enabled to mitigate risks.

3 Project Setup and Steps

3.1 Step 1: Setting Up GHIDRA

1. **Installation**: I installed GHIDRA and ensured JDK 21+ was configured, as GHIDRA requires this for its scripting functions.
2. **Project Creation**: A new project was created in GHIDRA to manage the imported malware samples, acting as a workspace for storing all analyses and modifications.

3.2 Step 2: Importing Malware Samples

1. Malware executables were imported into GHIDRA, and I ensured that antivirus protection was active due to the potentially dangerous nature of these files.

2. Each file was opened in GHIDRA's **Code Browser**, allowing for assembly code viewing and in-depth analysis.

3.3 Step 3: Opcode Extraction Using Python Script

To automate the extraction of opcodes, a Python script was developed, leveraging GHIDRA's scripting capabilities. The script iterates through each instruction within the malware's code and saves the opcode to a file. The Python script (`ghidra_opcode_script.py`) performs the following:

- Loads the current program in GHIDRA, targeting the instructions section.
- For each instruction in the program, retrieves the opcode and writes it to a file.
- The output filename is derived from the original malware sample name, saved with a `.opcode` extension (e.g., `malware_sample.opcode`).

3.4 Code Snippet

Here's an snippet from one of the python scripts $\texttt{ghidra}_{o}pcode_{s}cript.pyfile$:

```
import os
import csv
import logging
from ghidra.app.util.headless import HeadlessScript
from ghidra.program.model.address import AddressSet

# Get script arguments and determine the save folder
argv = getScriptArgs()

try:
    # Set save folder
    if len(argv) == 2:
        output_folder = argv[0]
        results_folder = argv[1]
    elif len(argv) == 1:
        output_folder = argv[0]
        results_folder = os.path.join(output_folder, 'results')
```

```
    elif len(argv) == 0:
        output_folder = os.getcwd()
        results_folder = os.path.join(os.getcwd(), 'results')
    else:
        raise ValueError("Invalid number of arguments")
except Exception as e:
    error_message = "An error occurred while setting parameters: {}".format(e)
    logging.error(error_message, exc_info=True)

program_name = currentProgram.getName()
program_folder = os.path.join(results_folder, program_name)

# Create the program-specific directory
if not os.path.exists(program_folder):
    os.makedirs(program_folder)

# Set up logging
log_file_path = os.path.join(output_folder, 'extraction.log')
logging.basicConfig(filename=log_file_path, level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')

# Determine file path for CSV file
csv_file_path = os.path.join(program_folder, program_name + '.csv')

try:
    with open(csv_file_path, 'w') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(['addr', 'opcode'])

        memory_blocks = currentProgram.getMemory().getBlocks()

        if memory_blocks:
            for block in memory_blocks:
                block_name = block.getName()
                address_set = AddressSet(block.getStart(), block.getEnd())
                instructions = currentProgram.getListing().getInstructions(address_set,
```

```
                for instr in instructions:
                    addr = instr.getAddress().toString()
                    opcode = str(instr).split(' ')[0]
                    csvwriter.writerow([addr, opcode])
        else:
            instructions = currentProgram.getListing().getInstructions(True)
            for instr in instructions:
                addr = instr.getAddress().toString()
                opcode = str(instr).split(' ')[0]
                csvwriter.writerow([addr, opcode])

except Exception as e:
    error_message = "An error occurred while writing the files: {}".format(e)
    logging.error(error_message, exc_info=True)
```

3.5 Step 4: Organizing and Saving Extracted Files

Each generated .opcode file was saved under a structured directory, with folder names reflecting the malware family. This organization facilitated easier access and ensured that each malware sample's opcodes were distinctly saved. Figure 1 shows the folder structure.

3.6 Step 5: GitHub Submission

The entire directory, including the extracted opcode files, the script used for extraction, and other relevant files, was archived into a .zip file and uploaded to our group's GitHub repository. The repository was kept private to protect sensitive data.

4 Challenges Encountered

- Antivirus Interference: Some malware samples triggered antivirus alerts. This was managed by carefully isolating the environment and maintaining antivirus scans.
- JDK Compatibility: GHIDRA required JDK 21+, necessitating a setup of the correct Java environment for GHIDRA's scripting functionalities.

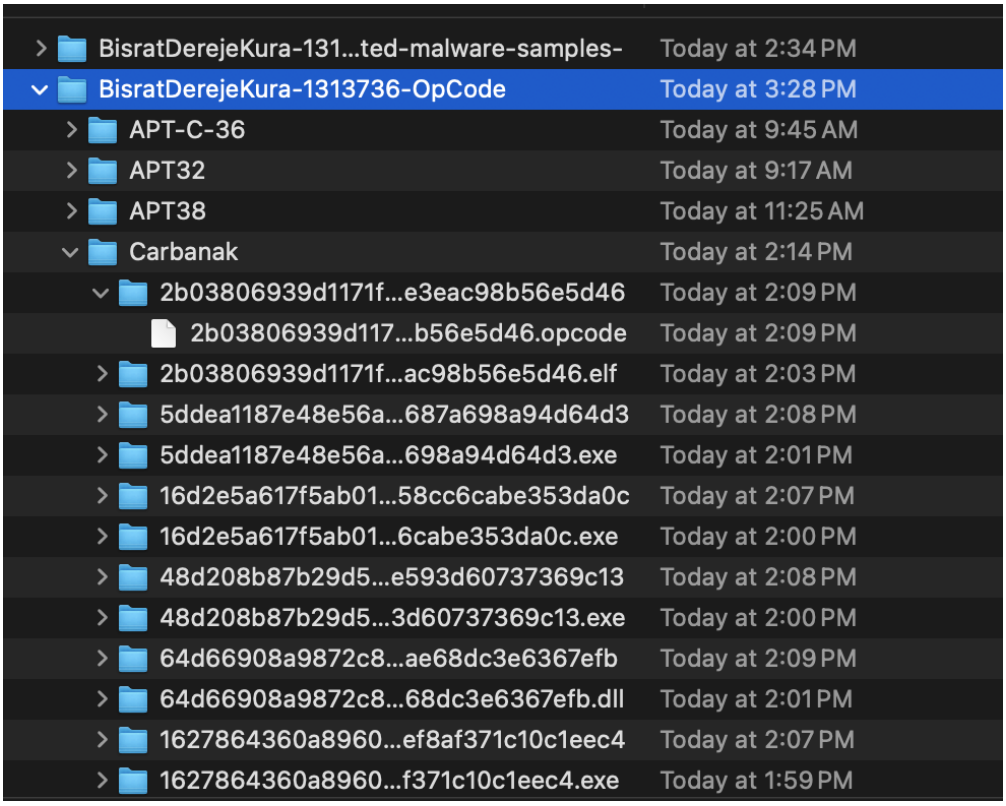

Figure 1: Folder structure for malware samples and extracted opcode files.

- File Naming Consistency: Initially, filenames were saved with the malware's original file extension. This was later adjusted to ensure only the base name was used, appending .opcode directly.

5 Conclusion

This project successfully automated the extraction of opcodes from malware samples using GHIDRA, with the extracted data providing valuable insights for malware pattern analysis and training AI models. GHIDRA's scripting capabilities proved highly efficient, allowing for systematic extraction of opcodes across numerous malware samples. By storing the extracted data in a structured GitHub repository, we achieved both data security and ease of access for further analysis.

6 Appendix

For reference, the following files were used in this project:

- ghidra_opcode_script.py - Python script for opcode extraction.

- `ExtractOpCodes.java - Java code for automating GHIDRA opcode extraction.`